The Journal of Systems and Software 212 (2024) 112019

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Bug priority change: An empirical study on Apache projects ☆

Zengyang Li [a], Guangzong Cai [a], Qinyi Yu [a], Peng Liang [b,*], Ran Mo [a], Hui Liu [c]

[a] School of Computer Science & Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, Central China Normal University, Wuhan, China
[b] School of Computer Science, Wuhan University, Wuhan, China
[c] School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China

ARTICLE INFO

Keywords:
Bug priority change
Open source software
Empirical study

ABSTRACT

In issue tracking systems, each bug is assigned a priority level (e.g., Blocker, Critical, Major, Minor, or Trivial in JIRA from highest to lowest), which indicates the urgency level of the bug. In this sense, understanding bug priority changes helps to arrange the work schedule of participants reasonably, and facilitates a better analysis and resolution of bugs. According to the data extracted from JIRA deployed by Apache, a proportion of bugs in each project underwent priority changes after such bugs were reported, which brings uncertainty to the bug fixing process. However, there is a lack of in-depth investigation on the phenomenon of bug priority changes, which may negatively impact the bug fixing process. Thus, we conducted a quantitative empirical study on bugs with priority changes through analyzing 32 non-trivial Apache open source software projects. The results show that: (1) 8.3% of the bugs in the selected projects underwent priority changes; (2) the median priority change time interval is merely a few days for most (28 out of 32) projects, and half (50. 7%) of bug priority changes occurred before bugs were handled; (3) for all selected projects, 87.9% of the bugs with priority changes underwent only one priority change, most priority changes tend to shift the priority to its adjacent priority, and a higher priority has a greater probability to undergo priority change; (4) bugs that require bug-fixing changes of higher complexity or that have more comments are likely to undergo priority changes; and (5) priorities of bugs reported or allocated by a few specific participants are more likely to be modified, and maximally only one participant in each project tends to modify priorities.

1. Introduction

Bug fixing is an important maintenance activity in the software development process. In issue tracking systems, each bug is assigned a priority (e.g., Blocker, Critical, Major, Minor, or Trivial in JIRA from the highest to lowest), which indicates the urgency level of the bug (Tian et al., 2016). In this sense, understanding bug priority changes helps to arrange the work schedule of participants reasonably, and better analyze and fix bugs. It can further clarify the roles of different participants in the bug fixing process, identify unreasonable behaviors of them, and ultimately standardize the bug fixing process.

Bugs with different priorities have different effects on software projects (Kononenko et al., 2016). Understanding priorities of bugs can help developers solve bugs in a proper manner (Zou et al., 2018). Therefore, it is important to assign and manage priorities of bugs appropriately. If the priorities of a substantial number of bugs are changed, it indicates delays in fixing critical bugs (Menzies and Marcus, 2008; Sharma et al., 2012; Chauhan and Kumar, 2020; Feng et al., 2012; Kumari and Singh, 2018). Deepening the understanding of bug priority changes can help solve serious bugs as soon as possible and avoid delays, identify areas that need tool support to automatically verify bug priority change requests, and better record these changes (Almhana et al., 2020).

According to the data extracted from issue tracking systems, a proportion of bugs underwent priority changes after they were reported. The changed priorities of those bugs may negatively impact on the bug fixing process. Hence, it is valuable to understand in depth the phenomenon of bug priority changes. In this work, we conducted an empirical study to investigate bug priority changes through analyzing the history and comments of bugs in issue tracking systems and related commits to the bugs.

Due to certain reasons in software development, the priority of a bug may change (Almhana et al., 2020; Gökçeoğlu and Sözer, 2021), and the trends of change of priorities of different bugs may be different. For instance, due to time pressure, the scheduled bug fixing tasks may

☆ Editor: Burak Turhan.

* Corresponding author.
E-mail addresses: zengyangli@ccnu.edu.cn (Z. Li), guangzongcai@mails.ccnu.edu.cn (G. Cai), qinyiyu@mails.ccnu.edu.cn (Q. Yu), liangp@whu.edu.cn (P. Liang), moran@ccnu.edu.cn (R. Mo), hliu@hust.edu.cn (H. Liu).

https://doi.org/10.1016/j.jss.2024.112019
Received 9 August 2023; Received in revised form 1 March 2024; Accepted 7 March 2024
Available online 14 March 2024
0164-1212/

be delayed, and hence, the priorities of the involved bugs are changed to lower priority levels (e.g., from Critical to Minor). In addition, some bugs may be found to result in severer consequences than expected; thus, the priorities of such bugs will be changed to higher priority levels (e.g., from Minor to Critical). The time when the priority change happens can be different for different bugs. The priority of a bug may be changed before bug fixing is started, or when bug fixing is in progress, and it can also be changed after a closed or resolved bug is reopened. Finally, the personal habits and styles of bug participants may also have an impact on priority changes. For instance, a bug participant may prefer to allocate a lower priority to a bug than other bug participants do.

Currently, there lacks a comprehensive understanding on bug priority changes, which may negatively impact the bug fixing process. To get a deep understanding on bug priority changes, we conducted an empirical study on 32 non-trivial Apache projects. The main contributions are summarized as follows:

- To the best of our knowledge, this work is a first attempt to explore the phases and patterns of bug priority changes.
- Twenty-four patterns of bug priority changes are identified to characterize the process of bug priority changes.
- We explored reasons for priority changes from the perspectives of the change complexity of bug-fixing commits and the communication complexity of bugs.
- We confirmed that the priority change is affected by several human factors.

The remaining of this paper is organized as follows. Section 2 presents the related work; Section 3 describes the design of the empirical study; Section 4 presents the results of the study; Section 5 discusses the study results; Section 6 identifies the threats to validity of the results; and Section 7 concludes this work with future research directions.

2. Related work

Most existing works regarding bug priority focus on the influence of bug priority on the development process or the prediction of bug priority, and there is only one existing study on bug priority changes. Therefore, in Section 2.1, we compare our study with the only research on bug priority changes; in Section 2.2, we discuss the related work on the role and impact of bug priority on the bug-fixing process; in Section 2.3, we present the recent studies on using machine learning or deep learning to predict bug priority.

2.1. Bug priority change

To our knowledge, only one existing work is focused on bug priority changes. Almhana et al. interviewed practitioners in industry and made quantitative analysis to understand the reasons why bug priorities are changed (Almhana et al., 2020). The reasons are summarized as follows: (1) the dependency of another bug's fix incorrect priority, (2) type/domain of project, (3) category of the bug report, (4) lack of time/heavy workload/tight schedule, (5) accident, (6) hot-fix request, and (7) business requirements.

Since the work of Almhana et al. (2020) is the only one that investigates bug priority changes, we compare the study results of their work with our current work. First, to study when the priority will change, Almhana et al. answered this question from the perspective of stakeholders' work schedules and project version release time; in contrast, we answered this question from the perspective of the bug life cycle, which helps participants adjust their work focus and schedule based on the bug life cycle. Second, Almhana et al. made an in-depth investigation on the causes of bug priority changes; in comparison, we provided a detailed description of the characteristics of bug priority changes, which is valuable for practitioners in analyzing bug priority and fixing bugs. Third, when investigating the human factors of bug priority changes, Almhana et al. studied the modification of priorities by stakeholders from the perspective of stakeholder types, while we focused on exploring whether there are specific participants who modify bug priorities in a different manner from others, which helps identify unreasonable behaviors during the bug fixing process. Finally, our study is the first attempt to link bug priority changes with code commits and team communications, which helps to gain a deeper understanding of the reasons for bug priority changes. In summary, we studied bug priority changes from various perspectives, which fills the knowledge gap in this field and expands our understanding of bug priority changes.

2.2. Impact of bug priority on the bug fixing

Bug priority is used in some studies to predict bug-fixing time. Akparinasaji et al. used bug priority to build the KNN model for predicting bug-fixing time (Akbarinasaji et al., 2018). Habayeb et al. used priority change as a factor to build a hidden Markov model to predict bug-fixing time (Habayeb et al., 2018). Vicira et al. included bug priority and other data fields to build a dataset for bug report evolution, and trained three machine learning models to estimate the bug-fixing time (Vieira et al., 2022). Yuan et al. mentioned that although bug priority and repair strategy may affect repair time, their experimental results showed that they had no necessary correlation (Yuan et al., 2020).

Some studies explored the impact of bug priority on bug-fixing process from different perspectives. Gavidia-Caldcron et al. pointed out that when the priority of an issue (which can be a bug) is raised to a level higher than its real assessment, it will hinder software development, and they used Game Theory to understand and fix the problem (Gavidia-Calderon et al., 2021). Motwani et al. indicated that automated Java repair techniques are moderately more likely to produce patches for high-priority bugs, while automated C repair techniques are not correlated with bug priority (Motwani et al., 2018). Etemadi et al. used bug priority to develop a scheduling-driven approach to effectively assign bug repair tasks to developers (Etemadi et al., 2021).

2.3. Bug priority prediction

Machine learning algorithms were widely used in the field of bug priority prediction. Jaweria et al. proposed and evaluated a priority recommendation approach based on classifiers using Naive Bayes and Support Vector Machine (SVM) (Kanwal and Maqbool, 2012). Alenezi et al. presented and evaluated an approach to predict the priority of a reported bug using Naive Bayes, Decision Trees, and Random Forest (Alenezi and Banitaan, 2013). Sharma et al. used multiple machine learning techniques such as SVM, Naive Bayes, K-Nearest Neighbors, and Neural Networks. In their results of cross-project validation, the accuracy of prediction of bug priority is above 70% except Naive Bayes (Sharma et al., 2012). Yuan et al. proposed a method to predict bug priority by machine learning, which takes advantage of several factors such as temporal, textual, author, related-report, severity, and product (Yuan et al., 2015).

Neural networks and deep learning have also been used in bug priority prediction in recent years. Yu et al. proposed neural network techniques to predict bug priority, adopted an evolutionary training process to solve error problems associated with new features, and reused datasets from similar software systems to accelerate the convergence of training (Yu et al., 2010). Kumari and Singh built improved classifiers using Naive Bayes and deep learning and considered measures such as severity, summary weight, and entropy to predict bug priority (Kumari and Singh, 2018). Izadi et al. proposed a two-stage approach to predict the priority level after the opening of an issue using feature engineering methods and state-of-the-art text classifiers (Izadi et al., 2022).

3. Study design

In order to investigate in depth the phenomenon of bug priority changes, we performed an empirical study on Apache Open Source Software (OSS) projects. In this section, we describe the empirical study, which was designed and reported following the guidelines proposed by Runeson and Höst (2009).

3.1. Objective and research questions

The goal of this study, described using the Goal-Question-Metric (GQM) approach (Basili, 1992), is to *analyze* the phenomenon of bug priority changes in depth, *from the point of view of* software developers *in the context of* OSS development. On the basis of the aforementioned goal, we have formulated five research questions (RQs), which are described as follows.

RQ1: What is the proportion of bugs with priority changes?
Rationale: With this RQ, we investigate the proportion of bugs with priority changes over all bugs of software projects, which gives practitioners and researchers a basic understanding on the state of bug priority changes.

RQ2: When is the bug priority changed?
Rationale: With this RQ, we investigate the phases when the priorities of bugs are changed after they are reported. Understanding the time trend of bug priority changes can help (a) practitioners arrange their work schedules reasonably, and (b) researchers build automated detection tools to identify unreasonable priority changes.

RQ3: How is the bug priority changed?
Rationale: This RQ is focused on investigating the number, pattern, trend, and range of bug priority changes, and whether they will be affected by the priority itself. Investigating the process of bug priority changes can help practitioners optimize existing bug report documents, making it easier for practitioners to understand bug reports and obtain more information related to bugs, and researchers can use the characteristics exhibited by the priority change process to improve prediction models related to bugs.

RQ4: Is there a significant difference between the complexity of bugs with priority changes and that of bugs without priority changes?
Rationale: With this RQ, we further explore the relationship between the priority changes of bugs and their change complexity of bug-fixing commits and communication complexity. The results of this RQ may partially reveal the reasons for bug priority changes from the perspective of workload and team communication.

RQ5: Do human factors play a role in bug priority changes?
Rationale: With this RQ, we study whether the bug priority change is related to different types of priority modifiers, whether priorities reported by specific participants or priorities allocated by specific participants are more likely to be modified, and whether specific participants tend to modify bug priorities. The result of this RQ can help (a) practitioners standardize the priority allocation process, and (b) researchers further investigate the roles of different participants in the bug fixing process.

3.2. Cases and unit analysis

This study investigates multiple OSS projects, i.e., cases, and each bug and its corresponding bug-fixing commit is a single unit of analysis.

3.3. Case selection

In this study, we only investigated Apache OSS projects. The reason is that the links between bugs and corresponding bug-fixing commits tend to be well recorded in the commit messages of those projects. For selecting each case (i.e., OSS project) included in our study, we applied the following criteria:

- C1: The five bug priority levels, i.e., Blocker, Critical, Major, Minor, and Trivial, are adopted to label the priority of each bug in the project.
- C2: The age of the project is more than 5 years.
- C3: The number of revisions (i.e., commits) of code repository of the project is more than 3000.
- C4: The number of bugs with priority changes in the project is more than 150.

Selection criterion C1 was set to ensure that the priority of each bug is explicitly defined. Criteria C2–C4 were set to ensure that the selected projects are non-trivial and the resulting dataset is big enough to be statistically analyzed. We selected all Apache projects (659 projects in total, and the initial list of projects have been made available in the replication package (Li et al., 2024)) and screened them, leaving only those that met the C1–C4 criteria, resulting in the final 32 projects. For example, *Beam* is excluded by C1, *Iceberg* is excluded by C2, *Zipkin* is excluded by C3, and *ZooKeeper* is excluded by C4.

3.4. Data collection

3.4.1. Data items to be collected

To answer the five RQs, we took a bug and its corresponding bug-fixing commit as the unit of analysis and the data items to be collected are shown in Table 1. Considering that data items D1 and D3 are straightforward, we only explain data items D2 and D4–D14 in detail.

D2: Priority. In JIRA, five priorities, Blocker, Critical, Major, Minor, and Trivial, are clearly defined (Apache, 2023) as follows:

- **Blocker**: a time-sensitive issue that is hindering a basic function of a project.
- **Critical**: a time-sensitive issue that is disrupting the project, but does not hinder basic functions.
- **Major**: this issue needs attention soon, but is not hindering basic functions. Most requests for new resources fall into this category.
- **Minor**: this issue needs attention, but is not time-sensitive and does not hinder basic functions.
- **Trivial**: this issue is minimal and has no time constraints.

D4: ChangePhase. We explain data item D4 (i.e., ChangePhase) in detail, since the task of collecting D4 is relatively complicated. We divide the period when the priority of a bug was changed into 4 phases: (1) **BEFORE**, if the priority of a bug was changed before the bug was handled; (2) **PROGRESS**, if the priority of a bug was changed during the process of handling the bug; (3) **REOPEN**, if the priority of a bug was changed after the bug was reopened; and (4) **AFTER**, if the priority of a bug was changed after the status of the bug was turned into "Resolved" or "Closed". Because the priority of a bug may be changed multiple times, we investigated the history of each bug to identify the change phase of each priority change. The procedure of labeling the ChangePhase of each bug priority change is defined as follows (also shown in Fig. 1):

- **S1**: Check whether the priority was changed after the status of the bug was changed to "Close" or "Resolve". If so, the ChangePhase of this priority change is labeled as AFTER.
- **S2**: Check whether the priority of the bug was changed after the bug was reopened. If so, the ChangePhase of this priority change is labeled as REOPEN.
- **S3**: If the status of the bug was changed to "In progress" or "Patch available" in the history of the bug, check whether priority was changed after bug status was changed to "In progress" or "Patch available".

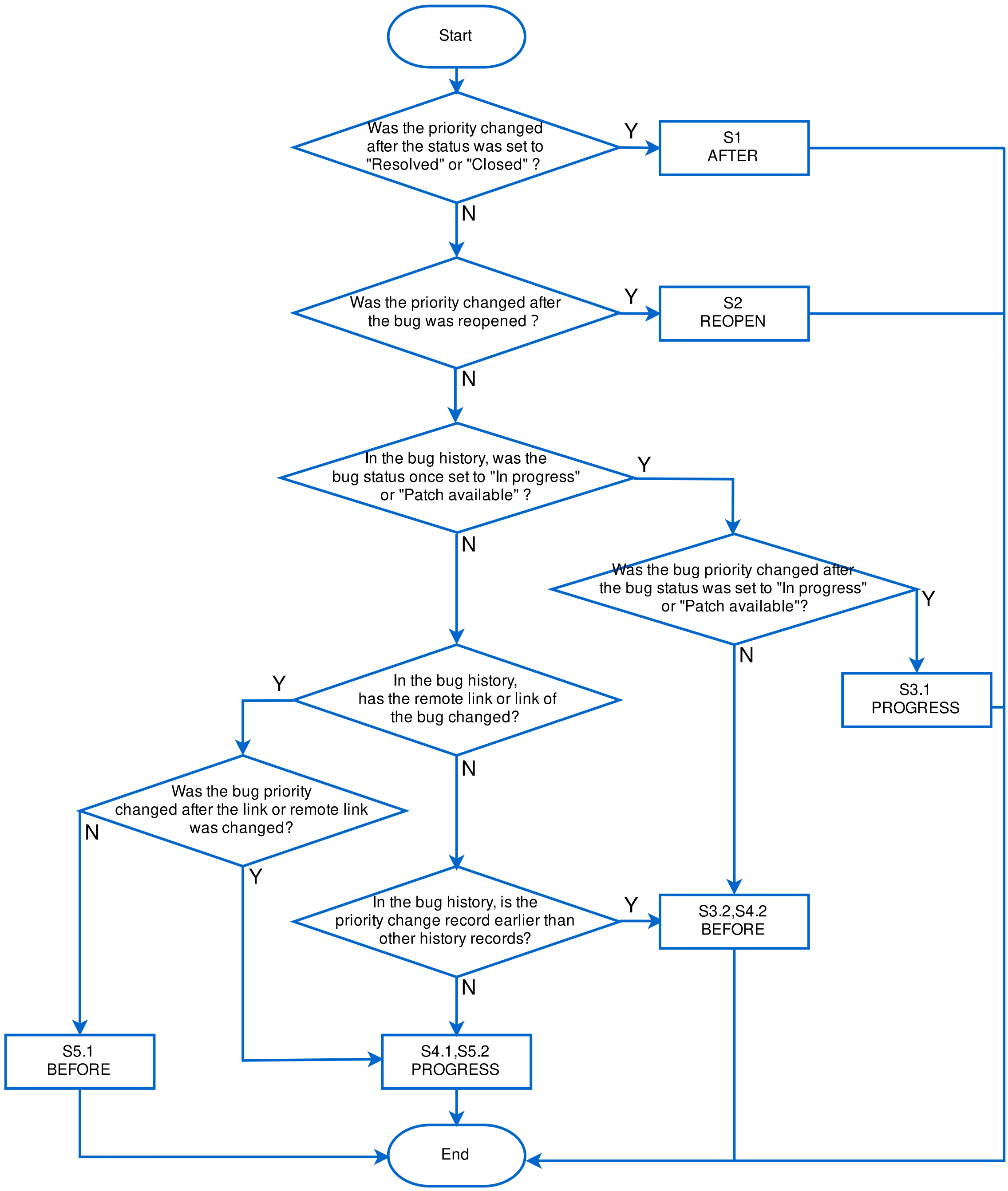

Fig. 1. Procedure of bug priority change phase identification.

- **S3.1**: If so, the ChangePhase of this bug priority change is labeled as PROGRESS.
- **S3.2**: If not, the ChangePhase of this priority change is labeled as BEFORE.

• **S4**: If the bug status has not changed to “In Progress” or “Patch Available” in the history of the bug, but its remote link or link has changed, check whether the priority of the bug was changed after its remote link or link was changed.

- **S4.1**: If so, the ChangePhase of this priority change is labeled as PROGRESS.
- **S4.2**: If not, the ChangePhase of this priority change is labeled as BEFORE.

• **S5**: If the bug status has not changed to “In Progress” or “Patch Available” in history of bug, and its remote link or link has not changed, check whether the priority change record is earlier than other history records.

- **S5.1**: If so, the ChangePhase of this priority change is labeled as BEFORE.
- **S5.2**: If not, the ChangePhase of this priority change is labeled as PROGRESS.

For each bug, we collected all its historical change items and arranged them in a chronological order (we provided five examples of historical information for five bugs in Fig. 2), and the historical information of a bug consists of multiple change records, each of which consists of the *Modifier*, *Change Time*, *Field Name*, *Original Value*, and

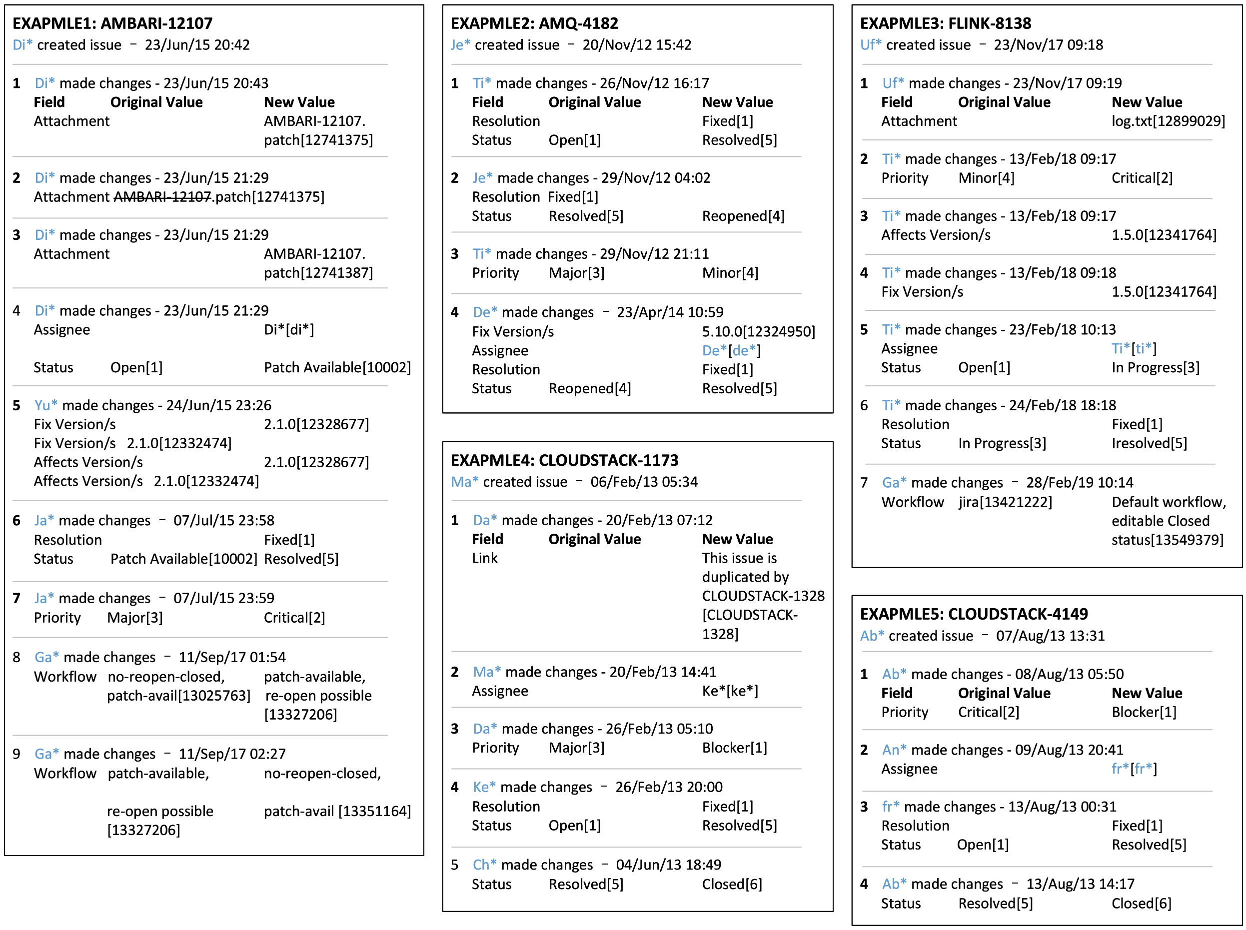

Fig. 2. Examples of historical information of five bugs.

Table 1
Data items to be collected for each bug and their mapping to the RQs.

#	Name	Description	RQ
D1	IsChanged	Whether priority of the bug was changed.	RQ1, RQ4
D2	Priority	The final priority level of a bug. Five priority levels are defined in JIRA: Blocker, Critical, Major, Minor, and Trivial, from the highest to the lowest level.	RQ2, RQ3
D3	ChangeInterval	The time interval between when the bug was reported and when priority was changed for the first time.	RQ2
D4	ChangePhase	The period of all priority changes of a bug. This item describes the periods of priority changes of a bug during its lifecycle.	RQ2, RQ3
D5	PrioritySequence	The sequence of changed priority levels. This item describes the changes of the priority of a bug during its lifecycle.	RQ3
D6	ChangePattern	The pattern of priority changes. This item describes the priority change pattern followed by the bug.	RQ3
D7	LOCM	The number of lines of code modified to fix a bug.	RQ4
D8	NOFM	The number of files (for Java) modified to a bug.	RQ4
D9	NOPM	The number of packages (for Java) modified to a bug.	RQ4
D10	Entropy	The normalized entropy of the modified source files for fixing a bug during the last 60 days (Li et al., 2020).	RQ4
D11	NOC	The number of comments on JIRA to a bug.	RQ4
D12	TLC	The total length of all comments to a bug (in bytes).	RQ4
D13	NOCR	The number of commenters to a bug.	RQ4
D14	PriorityModifier	The author of a priority change of a bug. There may be multiple PriorityModifiers for a bug since it may undergo multiple priority changes made by different practitioners.	RQ5

New Value. Then we calculated the ChangePhase for this bug priority change according to the above process (i.e., situation S1 to situation S5). The division of situation S1 and situation S2 is easy to understand (corresponding examples are shown in EXAMPLE1 and EXAMPLE2 in Fig. 2), and here we explain the reasons for dividing situation S3, situation S4, and situation S5. For some bugs, their historical status is displayed as "In progress" or "Patch available", indicating that such bugs have started to be handled. For example, the status of EXAMPLE3 in Fig. 2 was changed from "Open" to "In progress" at 10:13 on 23/Feb/18 (corresponding to change number 5). Prior to this status change, a priority change occurred at 09:17 on 13/Feb/18 (corresponding to change number 2). Therefore, we believe that this priority change corresponds to situation S3.2, which is "BEFORE". However, in the life cycle of some bugs, their status does not change to "In progress" or "Patch available", as the process of adjusting the bug status is not very rigorous. For example, the status of EXAMPLE4 and EXAMPLE5 in Fig. 2 were directly change from "Open" to "Resolved". One of the reasons is that the bug (e.g., EXAMPLE4 in Fig. 2) is linked to another bug, such as a duplicate bug. After fixing the linked bug, this bug is also closed. Therefore, if the status of a bug is not changed to "In progress" or "Patch available" but is linked to another bug, we believe

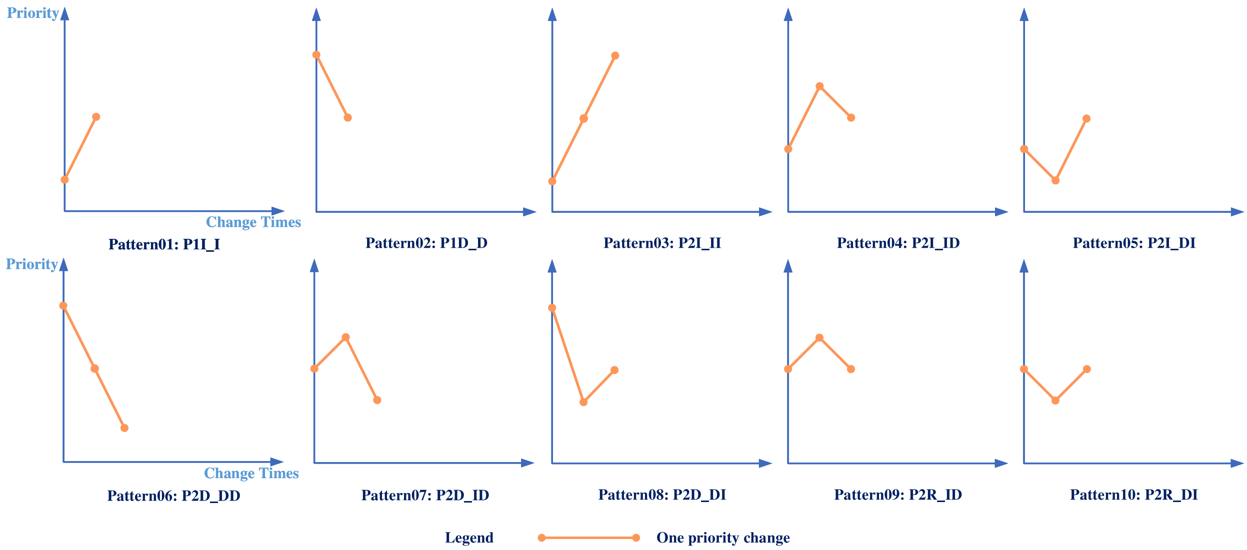

Fig. 3. Bug priority change patterns when a bug's priority is changed for one or two times.

that the bug begins to be handled when it is linked to another bug, such as EXAMPLE4 in Fig. 2. The priority change of the bug occurs after it is linked to another bug, so this priority change corresponds to situation S4.1, which is "PROGRESS". If the ChangePhase for a priority change cannot be determined after the above process, then we can only compare the priority change with other historical change records. We believe that in this case, as long as any field of a bug (except for "Priority") changes, it can be considered that the bug has started to be handled. For example, EXAMPLE5 in Fig. 2, the priority change occurred before any field was modified, corresponding to situation S5.1, which is "BEFORE".

D5: PrioritySequence. Different priority levels correspond to different numbers. Bug priority levels Blocker, Critical, Major, Minor, and Trivial in JIRA correspond to 1, 2, 3, 4, and 5 respectively. The PrioritySequence of a bug is recorded by a sequence of priority numbers in chronological order of priority changes. For instance, when the priority of a bug was changed from Blocker to Minor, and then from Minor to Major, the PrioritySequence is recorded as "143".

D6: ChangePattern. To investigate how the bug priority is changed, we summarize all possible ways of bug priority changes based on a 4-point model (the four points are initial priority, final priority, highest priority, and lowest priority of a bug) into a list of bug priority change patterns (a total of 24 change patterns), which are shown in Figs. 3 and 4. Compared with only considering the initial priority and final priority, we use four points to summarize all patterns of priority changes, which can reflect the priority change process in more detail, rather than just telling the final result of the priority changes. When we summarize the priority change patterns, we distinguish different number of times the priority changes. This is because the vast majority of bugs only undergo one or two priority changes, and the process of one or two priority changes is simpler. Therefore, we list all points in the process of one or two priority changes, and every two neighboring points are connected by a solid line in Fig. 3. Correspondingly, only a very small number of bugs undergo three or more priority changes, and the change process is relatively complicated. We use four points to describe the change process, and every two neighboring points are connected by a dotted line in Fig. 4. We have arranged a sequence number for each pattern and assigned an alias based on its changing characteristics. The alias for a pattern is in one of the following forms: PnC_X, PnC_XY, PnC_XYZ. The rules for taking aliases are described as follows: (1) *P* is the abbreviation of word "Pattern"; (2) letter *n* following letter *P* represents the number of priority changes for the bug: 1, 2, and *m* denote that the bug undergoes 1, 2, and more than 2 priority changes respectively; (3) letter *C* represents the final result of the priority changes for the bug: *I* indicates that the bug priority is changed to a higher level, *D* indicates that the bug priority is changed to a lower level, and *R* indicates that the bug priority returns to the original level; and (4) the string of letters after the underscore represents the process of bug priority changes.

Pattern01 ∼ Pattern02 and Pattern03 ∼ Pattern10 in Fig. 3 show all cases in which the bug priority changes for one or two times, respectively. Specifically, the alias of Pattern01 is P1I_I, and the fact that a bug's priority change pattern is P1I_I means that: this bug undergoes a priority change, its final priority is changed to a higher level, and the change process is that the priority is increased once. P1D_D can be interpreted in a similar way. In P2I_II, P2I_ID, and P2I_DI, after different change processes, the bug priority is changed to a higher level finally; in P2D_DD, P2D_ID, and P2D_DI, the bug priority is changed to a lower level finally; in P2R_ID and P2R_DI, the bug priority returns to the original level finally after a change to a higher or lower level.

In Fig. 4, all patterns show the cases in which the bug priority is changed three or more times. We have summarized 14 patterns according to the initial priority, the final priority, the highest priority, and the lowest priority of a bug. In Pattern11 ∼ Pattern15 (i.e., PmI_I, PmI_ID, PmI_DI, PmI_IDI, PmI_DID), after ups and downs the bug priority is changed to a higher level finally. In Pattern16 ∼ Pattern20 (i.e., PmD_D, PmD_ID, PmD_DI, PmD_IDI, PmD_DID), after ups and downs the bug priority is changed to a lower level finally. In Pattern21 ∼ Pattern24 (i.e., PmR_ID, PmR_DI, PmR_IDI, PmR_DID), after ups and downs the bug priority returns to the original level. Each bug with priority changes can be labeled as one of the 24 bug priority change patterns.

D7 ∼ D13: LOCM, NOFM, NOPM, Entropy, NOC, TLC and NOCR. We chose D7 ∼ D10 as indicators for change complexity of bug-fixing commits, as these indicators have been widely used in previous studies (Oliveira et al., 2020; Najafi et al., 2019; Al-Sabbagh et al., 2022; Li et al., 2020). D11 ∼ D13 were chosen as indicators of the communication complexity of a bug since these three indicators are directly obtainable information, and they are simple, easy to understand, and can intuitively represent the communication complexity of a bug. D7 ∼ D9 and D11 ∼ D13 are clearly defined, and here we only explain the definition of the entropy of the modified source files in a commit (i.e., D10) in detail (Hassan, 2009). Suppose that the modified source files of commit c are $\{f_1, f_2, \ldots, f_n\}$, and file $f_i\,(1 \le i \le n)$ has been modified m_i times (i.e., in m_i commits) during a period of time before this commit. Let

$$p_i = m_i / \sum_{i=1}^{n} m_i. \tag{1}$$

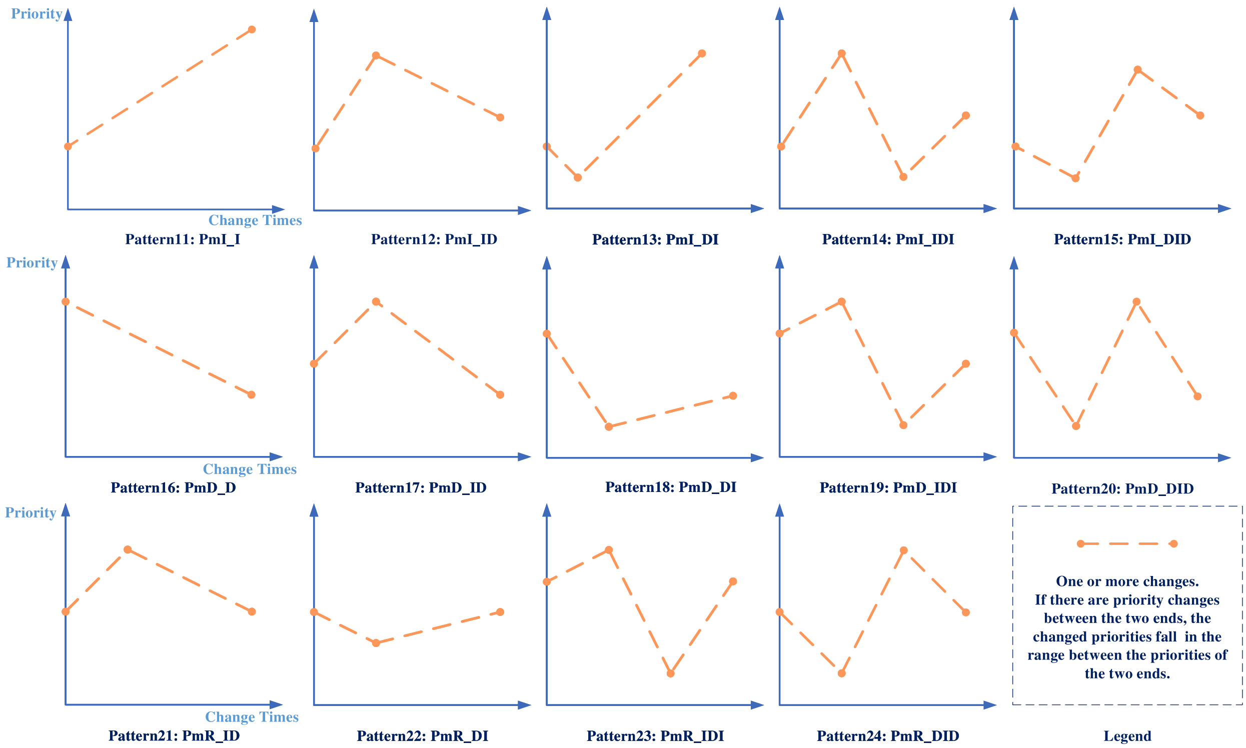

Fig. 4. Bug priority change patterns when a bug's priority is changed for three or more times.

Then, the entropy

$$H(n) = -\sum_{i=1}^{n} p_i log_2 p_i. \tag{2}$$

The normalized entropy

$$\tilde{H}(n) = \begin{cases} H(n)/log_2 n & \text{n} > 1, \\ 0 & \text{n} = 1. \end{cases} \tag{3}$$

In this study, the period is set to 60 days (including the day when commit c happened), which is chosen according to the period set by Hassan and Li et al. when calculating the entropy of the modified source files for fixing a bug (Hassan, 2009; Li et al., 2020).

D14: PriorityModifier. People related to a bug are referred to as participants. We divide participants into five types: Reporter, Assignee, Commenter, FieldModifier, and Other. PriorityModifier can be these five types of participants. Reporter denotes the participant who reports the bug; Assignee denotes the participant to whom the bug is assigned; Commenter denotes the participant who makes comments on this bug; FieldModifier denotes the participant who modifies fields of the bug except for the field of priority; Other denotes the participant who does not preform any of the above activities.

3.4.2. Data collection procedure

We have formulated the following rules in advance to filter and process the priority change records that we have collected.

- **R1**: Check whether the original priority of the bug or the priority of the bug was changed is null. If so, this situation is not considered as a priority change of the bug.
- **R2**: Check whether the priority of the bug was changed by the same PriorityModifier within 5 min after the bug was reported. If so, this situation is not considered as a priority change, but the priority after change is considered as the initial priority. For example, when a bug is created, the priority is Major. One minute later, the same PriorityModifier changes the priority to Critical. Then we do not consider this priority modification as a priority change, and treat the priority of this bug when created as Critical.
- **R3**: Check whether the bug priority is changed back to the original priority by the same PriorityModifier within 5 min after the priority is modified. If so, these two priority modifications do not count.
- **R4**: Check whether the priority of the bug is modified by the same PriorityModifier within 5 min after the priority is modified, but it is not changed back to the original priority. If so, the two priority changes are combined into one, that is, from the original priority to the final priority. The time of change is subject to the time of the second change. For example, a PriorityModifier changed the priority of a bug from Major to Critical at 18:00 on October 26, 2020, and the same PriorityModifier changed the priority from Critical to Blocker at 18:01 on October 26, 2020. Then we merge the two priority modification records, that is, this PriorityModifier changed the bug priority from Major to Blocker at 18:01 on October 26, 2020.

R1 is formulated to filter the situation where the priority is null. **R2**, **R3**, and **R4** are formulated to deal with the problem of multiple modifications by the same PriorityModifier in a short time for personal reasons, such as regret. This time period is set as 5 min. The reason is that we manually checked the multiple modifications in a short period of time, and found that the modifications after the creation of a bug or the modification of the priority are mainly made in the next 5 min, while the modifications after 5 min are scattered.

The procedure of collecting the data items listed in Table 1 consists of seven steps, as shown in Fig. 5. For each selected project, the details of the steps are described as follows.

- **Step 1**: Extract bugs from JIRA. According to the APIs[1] provided by JIRA, we developed a tool to extract issues. We used this tool to obtain all issues of Apache OSS Projects.
- **Step 2**: Filter out priority change records. According to R1, we filtered out the records with null priority.

[1] https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/.

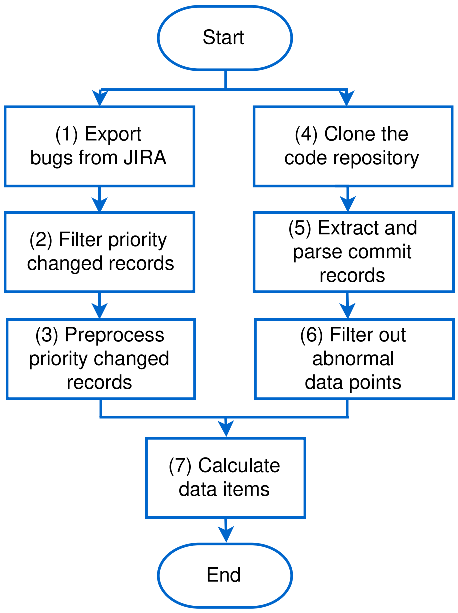

Fig. 5. Procedure of data collection.

- **Step 3**: Preprocess priority change records. According to R2-R4, we processed multiple priority changes made by the same PriorityModifier in a short time.
- **Step 4**: Clone the code repository. With the TortoiseGit tool, we cloned the Git repository of the project.
- **Step 5**: Extract and parse commit records. First, we removed the duplicate commit records introduced for unknown reasons and only retained one of the duplicate commit records. We found that there may be duplicate commit records in which all the change details are identical except for the revision ID. Second, we extracted bug ID from the message in each commit record and mapped each bug and commit record according to bug ID. Because not all the bugs in issues can be mapped to commits, the bugs after parsing the commits (i.e., the bugs that can be mapped to commits) are less than bugs in JIRA.
- **Step 6**: Filter out abnormal data points. We found that some bugs which fixing involves either more than 100 modified source files or over 10,000 lines of modified code (Li et al., 2020). These data items can affect the validity of our conclusions.
- **Step 7**: Calculate data items. We counted the priority change records of each bug and calculated D1–D5. By analyzing the PrioritySequence (i.e., D5) of the bug, we can calculate the ChangePattern (i.e., D6) of each bug with priority changes. D7–D10 were calculated based on related commit records. D11 is an inherent field of each bug, which can be directly obtained. Finally, we collected PriorityModifier (i.e., D12) of each bug.

3.5. Data analysis

The answers to RQ1, RQ2, and part of RQ3 can be obtained by descriptive statistics. Here, we only describe the calculation process of bug priority change range for RQ3, and details of data analysis for RQ4 and RQ5.

Priority change range. To answer RQ3 about the priority change range, we calculate the average priority change range of the bug according to the PrioritySequence of each bug. For example, if the PrioritySequence of a bug is 143, the priority change range is $(|4-1|+|3-4|)/2 = 2$.

Change complexity of bug-fixing commits and communication complexity. The change complexity of bug-fixing commits of a bug is indicated by LOCM, NOFM, NOPM, and Entropy. Communication complexity is indicated by NOC, TLC, and NOCR. To answer RQ4, in addition to descriptive statistics, we performed Mann–Whitney U tests (Field, 2013) to examine if two groups are significantly different from each other. Bugs are divided into two types: bugs with priority changes and bugs without priority changes. We calculated the indicators of change complexity of bug-fixing commits and the indicators of communication complexity for these two types of bugs, and then used Mann–Whitney tests to examine whether there is significant difference between the two types of bugs with respect to each indicator. Since the variables to be tested do not necessarily follow a specific distribution, it is reasonable to use the Mann–Whitney U test – a non-parametric test – in this study. The test is significant at *p-value* < 0.05, which means that the tested groups have a significant difference. In addition, we also calculated the effect size (*Cohen'd*) to show the difference in size between the two sets of data.

Types of PriorityModifier. A PriorityModifier may belong to multiple types, for example, a PriorityModifier may be both the Reporter and Assignee of a bug, and has also made comments on the bug. We calculated the proportion and distribution of different types of PriorityModifier respectively. When calculating the proportion of different types of PriorityModifier, each PriorityModifier can belong to multiple types if applicable. When calculating the distribution of different types of PriorityModifier, each PriorityModifier only belongs to one type. The meaning of PriorityModifier type here changed, and two new PriorityModifier types were added. Their meanings are described as follows: Reporter denotes the person who reported the bug but is not the one assigned to the bug; the meaning of Assignee is opposite to that of Reporter; Reporter&Assignee denotes the reporter and assignee of the bug; Commenter denotes the person who only makes comments when she is not a Reporter or Assignee; FieldModifier denotes the participant who only modifies fields of the bug except for the Priority field when the participant is not a Reporter or Assignee; Commenter&FieldModifier denotes the person who makes comments and modifies fields of the bug except for the Priority field when she is not a Reporter or Assignee; Other denotes the participant who did not perform any of the above activities.

Bug participants prone to making priority changes. To study the possible human factors in bug priority changes, we investigated three specific groups of bug participants: (1) participants whose reporting bug priorities are likely to be modified, denoted as PRs; (2) participants whose allocating bug priorities are likely to be modified, denoted as PAs; and (3) participants who are likely to modify bug priorities, denoted as PMs. First of all, according to RQ1, we can get the overall bug priority change probability of all projects (denoted as p). Then, we calculate the average number of bugs reported (denoted as N_r), average number of priorities allocated (i.e., the number of priorities modified, denoted as N_a), and average number of bugs involved by each core participant (denoted as N_c) in all projects. According to the work of Cheng et al. (2017), supposing that the participants are ranked by the number of their involved bugs in a descending order in each project, core participants are the top participants whose total number of involved bugs reaches 80% of the total number of bugs in the project. Definitions of PR, PA, and PM are described as follows: (1) If a participant reports at least N_r bugs, and these bugs have their priority modified later with the probability of at least $2p$, then the participant is a PR; (2) If there are at least N_a priorities allocated to a participant and these priorities will be modified with the probability of at least $2p$, then the participant is a PA; (3) If the ratio between the number of priorities modified by a participant and the number of bugs she involves in is at least $2p$, then the participant is a PM. The reason why we set the priority change probability at $2p$ here is that we regard that more than twice the overall priority change probability (i.e., p) is a higher priority change probability, while N_r, N_a, and N_c are to ensure sufficient bug samples for each participant.

Table 2
Demographic information of selected Apache OSS projects.

Project	Age(y)	Language	Domain	#Revision	#Committer	#Bug in JIRA
ActiveMQ	17	Java	Message Middleware	10,728	166	5,058
Ambari	11	Java, JavaScript, Python	Hadoop Clusters Administration Tool	24,110	236	17,961
Arrow	8	C++, Java, Go, Python	Columnar Memory Format	12,327	948	5,576
Axis2	18	Java, C, C++	Web Service Container	14,030	87	4,558
Camel	16	Java	Integration Framework	55,632	1100	5,758
CloudStack	12	Java, Python	Infrastructure-as-a-Service Platform	32,759	608	7,861
Cordova	14	JavaScript, Java, Objective-C	Mobile App Development Platform	33,010	1223	8,780
Drill	10	Java	SQL Query Engine	4,313	218	5,437
Flink	12	Java, Scala	Distributed Processing Engine	31,286	1598	11,003
Geode	12	Java	Data Management Platform	10,803	204	5,429
Groovy	19	Java, Groovy	Dynamic Language	18,983	440	6,921
Guacamole	12	C, Java, JavaScript	Clientless Remote Desktop Gateway	8,069	141	856
Hadoop	13	Java	Distributed Processing Framework	25,738	645	23,798
HBase	16	Java	Distributed Database	19,428	600	12,227
Hive	14	Java	Data Warehouse	16,040	487	14,306
Hudi	6	Java, Scala	Data Lake	3,314	381	1,177
Ignite	9	Java, C#, C++	Distributed In-Memory Computing Platform	22,146	456	7,242
Impala	11	C++, Java, Python	Distributed SQL Query Engine	10,379	243	6,238
Jackrabbit-Oak	11	Java	Hierarchical Content Repository	17,839	77	3,726
Kafka	11	Java, Scala	Streaming Message Middleware	10,361	1116	6,808
Lucene	21	Java	Full-text Search Library	35,469	386	4,201
Mesos	12	C++	Cluster Manager	18,175	388	4,922
NetBeans	9	Java	Integrated Development Environment	5,320	235	4,740
NiFi	8	Java	Processing Engine	7,303	570	4,427
OFBiz	16	Java, JavaScript	Java Web Framework	32,911	89	4,740
Ozone	5	Java	Distributed Key-Value Store	4,830	193	2,471
Qpid	16	Java, C++	Message Queue	45,216	121	5,097
Solr	21	Java	Enterprise Search Platform	32,002	228	7,209
Spark	13	Scala, Python, Java	Distributed Processing Engine	32,561	2572	15,508
Thrift	16	C++, Java, C	Cross-Language Services Development	6,553	562	3,058
Traffic Server	13	C++, Python, C	Caching Proxy Server	13,643	381	3,026
Wicket	18	Java	Java Web Framework	21,108	142	4,193

4. Study results

We collected data items described in Table 1 from 32 non-trivial Apache OSS projects that were selected following the criteria set defined in Section 3.3. The data of the selected projects were collected around the beginning of September 2022. The replication package of this study has been made available online (Li et al., 2024), including the raw data, code, calculation results, and a README file. The demographic information of the 32 projects is shown in Table 2. The age of each project falls in the range from 5 to 21 years, the number of revisions (i.e., commits) of each project falls in the range between 3314 and 55,632, the number of committers of each project falls in the range between 77 and 2572, and the number of bugs reported in JIRA falls in the range between 856 and 23,798. In the rest of this section, we present the results for each RQ.

4.1. Proportion of bugs with priority changes (RQ1)

Table 3 shows the proportion of bugs with priority changes for each selected project and that for all selected projects as a whole. In this table, #BugPC denotes the number of bugs with priority change(s). Specifically, the proportion of bugs with priority changes for each selected project ranges from 1.7% (project *Ambari*) to 24.4% (project *Guacamole*), and when taking all projects as a whole, the proportion of bugs with priority changes is 8.3%, which is relatively low.

4.2. Time characteristics of bug priority changes (RQ2)

4.2.1. Time intervals of bug priority changes

We first investigated the time intervals between when the bugs were reported and when their priorities were changed for the first time, and the result is shown in Table 4. In each project, the minimum time interval of bugs ranges from 0.0001 to 0.0040 days, i.e., 9 to 346 s, and the average time interval of bugs ranges from 14.83 to 211.36 days, while the median time interval ranges from 0.16 to 15.07 days, which is

Table 3
Proportion of bugs with priority changes (RQ1).

Project	#Bug in JIRA	#BugPC	%
ActiveMQ	5,058	216	4.3%
Ambari	17,961	307	1.7%
Arrow	5,576	290	5.2%
Axis2	4,554	425	9.3%
Camel	5,758	807	14.0%
CloudStack	7,861	870	11.1%
Cordova	8,780	813	9.3%
Drill	5,437	579	10.6%
Flink	10,296	2,049	19.9%
Geode	5,429	150	2.8%
Groovy	6,918	352	5.1%
Guacamole	856	209	24.4%
Hadoop	23,798	2,244	9.4%
HBase	12,227	1,142	9.3%
Hive	14,306	509	3.6%
Hudi	1,177	184	15.6%
Ignite	7,242	582	8.0%
Impala	6,238	1,431	22.9%
Jackrabbit-Oak	3,726	168	4.5%
Kafka	6,808	617	9.1%
Lucene	4,201	201	4.8%
Mesos	4,922	427	8.7%
NetBeans	4,740	314	6.6%
NiFi	4,427	221	5.0%
OFBiz	4,740	215	4.5%
Ozone	2,471	194	7.9%
Qpid	5,097	214	4.2%
Solr	7,209	377	5.2%
Spark	15,265	1,773	11.6%
Thrift	3,058	183	6.0%
Traffic Server	3,026	165	5.5%
Wicket	4,193	212	5.1%
All projects	**223,355**	**18,440**	**8.3%**

Table 4
Time intervals (in days) of bug priority changes (RQ2).

Project	Min	Median	Average	Max	365 days # (%)
ActiveMQ	0.0022	10.95	125.10	1440.44	17 (7.9%)
Ambari	0.0005	0.29	14.83	1258.75	1 (0.3%)
Arrow	0.0007	0.85	46.77	883.01	10 (3.4%)
Axis2	0.0007	15.07	72.34	2280.70	12 (2.8%)
Camel	0.0004	0.50	19.66	878.04	11 (1.4%)
CloudStack	0.0013	5.93	30.07	1028.01	3 (0.3%)
Cordova	0.0010	5.13	54.25	977.14	33 (4.1%)
Drill	0.0007	12.92	49.38	998.90	12 (2.1%)
Flink	0.0002	8.20	180.29	2452.13	280 (13.7%)
Geode	0.0001	5.20	170.92	1157.41	29 (19.3%)
Groovy	0.0004	4.06	145.06	4024.97	34 (9.7%)
Guacamole	0.0005	0.16	16.49	334.85	0 (0.0%)
Hadoop	0.0006	2.92	65.17	2163.18	114 (5.1%)
HBase	0.0004	1.65	49.60	2774.03	41 (3.6%)
Hive	0.0020	4.76	61.10	1349.05	24 (4.7%)
Hudi	0.0010	6.74	64.35	1007.95	13 (7.1%)
Ignite	0.0001	4.95	94.41	1294.59	54 (9.3%)
Impala	0.0001	13.07	113.77	2611.83	145 (10.1%)
Jackrabbit-Oak	0.0040	3.27	63.69	1533.01	7 (4.2%)
Kafka	0.0003	1.75	64.45	2372.47	31 (5.0%)
Lucene	0.0015	2.84	211.36	2028.66	42 (20.9%)
Mesos	0.0008	5.07	90.36	1189.66	35 (8.2%)
NetBeans	0.0013	7.24	61.79	1184.81	14 (4.5%)
NiFi	0.0002	0.85	33.21	978.16	5 (2.3%)
OFBiz	0.0004	2.46	108.93	2144.06	22 (10.2%)
Ozone	0.0014	3.44	55.37	1246.04	8 (4.1%)
Qpid	0.0003	4.07	45.29	843.26	8 (3.7%)
Solr	0.0005	4.64	81.78	2122.48	24 (6.4%)
Spark	0.0001	0.92	27.31	1453.73	28 (1.6%)
Thrift	0.0035	7.73	171.36	2642.75	25 (13.7%)
Traffic Server	0.0006	3.15	56.11	944.63	6 (3.6%)
Wicket	0.0007	0.30	20.31	1652.07	1 (0.5%)

way shorter than the average. In addition, the median time interval of priority changes is merely a few days for all projects except for projects *ActiveMQ*, *Axis2*, *Drill* and *Impala*.

As shown in Table 4, the time intervals of priority changes of some bugs are more than 365 days. For most (19 out of 32) of projects, the time intervals of less than 5.0% of the bugs are more than 365 days. For 6 of the projects, the time intervals of more than 10.0% of the bugs are more than 365 days.

4.2.2. Change phases of bug priority

The distribution of change phases of bug priority changes is shown in Table 5. For 75% of (24 out of 32) projects, bug priority changes that happened in change phase BEFORE (i.e., before the bug was handled) account for the most. For the remaining projects (8 out of 32), bug priority changes that happened in change phase PROGRESS (i.e., during the period when the bug was being handled) account for the most. In addition, for all the projects, only a very small proportion of bug priority changes were made in change phases REOPEN (i.e., after the bug was reopened) and AFTER (i.e., after the bug's status was set as Closed or Resolved) respectively.

4.3. Patterns of bug priority changes (RQ3)

In this subsection, we first calculate the number of priority changes of bugs in each project, and then investigate the priority change patterns of bugs in each project. Finally, we show the trend of priority changes and the distribution of the average change range of each bug, and explore the relationship between priority change and priority itself.

4.3.1. Number of priority changes of bugs

Table 6 shows the distribution of bugs over the number of priority changes for each project. For all projects, most (at least 70.7%) bugs underwent only one priority change, and a small proportion (no more than 21.2%) of bugs underwent two priority changes. Additionally, a small proportion of bugs underwent three or more priority changes in 31 out of the 32 projects, and the other project (i.e., *Geode*) does not have any bugs with three or more priority changes. When taking all projects as a whole, 87.9% of the total bugs underwent only one priority change, 9.8% of the total bugs underwent two priority changes, 1.8% of the total bugs underwent three priority changes, and only 0.5% of the total bugs underwent more than three priority changes.

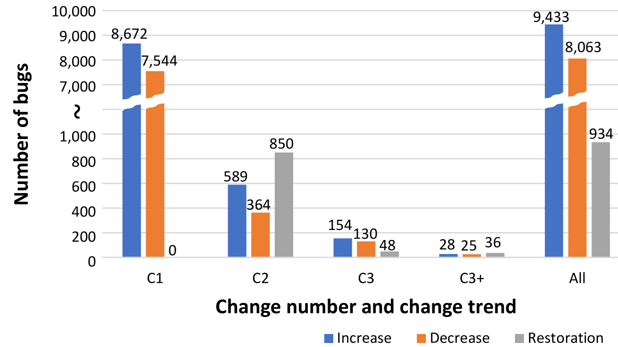

Fig. 6. Distribution of the number of bugs over priority change trend and the number of changes (RQ3).

4.3.2. Bug priority change patterns

We consider all possible cases of priority changes of a bug and propose 24 priority change patterns (shown in Figs. 3 and 4) of a bug. The distribution of the number of bugs of each project over different priority change patterns is shown in Table 7, in which each cell lists the number and percentage of bugs with the priority change pattern in the corresponding row. The cell in the far right column lists the number of patterns for each project, and the last row lists the number of projects for each pattern.

As shown in Table 7, the distribution of bugs over priority change patterns are quite different from project to project. Specifically, from the perspective of the coverage of the bug priority change patterns (i.e., the rightmost column in Table 7), none of the projects covers all the bug priority change patterns, and the number of bug priority change patterns for all the projects falls in the range between 6 (projects *Geode* and *Traffic Server*) and 22 (projects *Flink* and *Impala*).

As shown in the second to last row of Table 7, over the 18,440 bugs with priority changes in total of all the projects, there are 8672 (47.5%) and 7544 (40.9%) bugs following priority change patterns P1I_I and P1D_D, respectively, which are way more than bugs with other patterns.

The last row in Table 7 shows the number of projects covering each bug priority change patterns. In the last row, P1I_I, P1D_D, and P2R_ID are covered by all projects, 8 patterns (i.e., P2I_II, P2I_ID, P2D_DD, P2D_ID, P2R_DI, PmI_I, PmI_ID, and PmD_D) are covered by more than a half of the projects, 3 patterns (i.e., PmI_IDI, PmD_IDI, and PmD_DID) are covered by only one project.

4.3.3. Bug priority change trend and range

Fig. 6 shows the trend of bugs with different numbers of priority changes. Priorities of 9443 out of 18,440 (51.2%) bugs increase finally; priorities of 8063 out of 18,440 (43.7%) bugs decrease finally; and priorities of 934 out of 18,440 (5.1%) bugs restore finally. In addition, 850 of 934 (91.0%) bugs that priority restore finally come from bugs that undergo two priority changes.

Fig. 7 shows the distribution of average priority change ranges of each bug for all projects (i.e., 18,440 bugs). For most of the bugs, their average priority change ranges are 1.0; the average priority change ranges of a minority of bugs are 2.0; and bugs with an average priority change ranges of 1.5 or 3.0 are rare.

Table 5
Distribution of change phases of bug priority changes (RQ2).

Project	Total	BEFORE # (%)	PROGRESS # (%)	REOPEN # (%)	AFTER # (%)
ActiveMQ	231	101 (43.7%)	**114 (49.4%)**	13 (5.6%)	3 (1.3%)
Ambari	317	97 (30.6%)	**163 (51.4%)**	13 (4.1%)	44 (13.9%)
Arrow	316	**227 (71.8%)**	78 (24.7%)	3 (0.9%)	8 (2.5%)
Axis2	503	189 (37.6%)	**277 (55.1%)**	33 (6.6%)	4 (0.8%)
Camel	867	**433 (49.9%)**	374 (43.1%)	13 (1.5%)	47 (5.4%)
CloudStack	1,027	417 (40.6%)	**463 (45.1%)**	111 (10.8%)	36 (3.5%)
Cordova	916	**488 (53.3%)**	388 (42.4%)	28 (3.1%)	12 (1.3%)
Drill	644	**332 (51.6%)**	294 (45.7%)	17 (2.6%)	1 (0.2%)
Flink	2,629	**1544 (58.7%)**	767 (29.2%)	130 (4.9%)	188 (7.2%)
Geode	158	**71 (44.9%)**	66 (41.8%)	18 (11.4%)	3 (1.9%)
Groovy	381	**181 (47.5%)**	176 (46.2%)	19 (5.0%)	5 (1.3%)
Guacamole	226	**171 (75.7%)**	49 (21.7%)	6 (2.7%)	0 (0.0%)
Hadoop	2,507	**1262 (50.3%)**	1,030 (41.1%)	70 (2.8%)	145 (5.8%)
HBase	1,272	**615 (48.3%)**	535 (42.1%)	50 (3.9%)	72 (5.7%)
Hive	549	**264 (48.1%)**	254 (46.3%)	15 (2.7%)	16 (2.9%)
Hudi	258	**134 (51.9%)**	114 (44.2%)	6 (2.3%)	4 (1.6%)
Ignite	674	**349 (51.8%)**	301 (44.7%)	11 (1.6%)	13 (1.9%)
Impala	1,792	**872 (48.7%)**	826 (46.1%)	72 (4.0%)	22 (1.2%)
Jackrabbit-Oak	181	**82 (45.3%)**	68 (37.6%)	7 (3.9%)	24 (13.3%)
Kafka	701	**447 (63.8%)**	199 (28.4%)	31 (4.4%)	24 (3.4%)
Lucene	213	86 (40.4%)	**111 (52.1%)**	10 (4.7%)	6 (2.8%)
Mesos	510	**270 (52.9%)**	215 (42.2%)	1 (0.2%)	24 (4.7%)
NetBeans	377	164 (43.5%)	**204 (54.1%)**	7 (1.9%)	2 (0.5%)
NiFi	239	**145 (60.7%)**	74 (31.0%)	11 (4.6%)	9 (3.8%)
OFBiz	238	107 (45.0%)	**108 (45.4%)**	13 (5.5%)	10 (4.2%)
Ozone	209	**141 (67.5%)**	59 (28.2%)	1 (0.5%)	8 (3.8%)
Qpid	230	**104 (45.2%)**	95 (41.3%)	8 (3.5%)	23 (10.0%)
Solr	413	**217 (52.5%)**	165 (40.0%)	18 (4.4%)	13 (3.1%)
Spark	2, 028	**965 (47.6%)**	819 (40.4%)	62 (3.1%)	182 (9.0%)
Thrift	202	85 (42.1%)	**82 (45.5%)**	5 (2.5%)	20 (9.9%)
Traffic Server	179	**94 (52.5%)**	74 (41.3%)	4 (2.2%)	7 (3.9%)
Wicket	228	**104 (45.6%)**	95 (41.7%)	12 (5.3%)	17 (7.5%)
All projects	21,215	**10,758 (50.7%)**	8,647 (40.8%)	818 (3.9%)	992 (4.7%)

Table 6
Distribution of bugs over the number of priority changes (RQ3).

Project	Total	C1 # (%)	C2 # (%)	C3 # (%)	C3+ # (%)
ActiveMQ	216	**205 (94.9%)**	9 (4.2%)	1 (0.5%)	1 (0.5%)
Ambari	307	**299 (97.4%)**	6 (2.0%)	2 (0.7%)	0 (0.0%)
Arrow	290	**268 (92.4%)**	19 (6.6%)	2 (0.7%)	1 (0.3%)
Axis2	425	**360 (84.7%)**	56 (13.2%)	6 (1.4%)	3 (0.7%)
Camel	807	**767 (95.0%)**	27 (3.3%)	11 (1.4%)	2 (0.2%)
CloudStack	870	**745 (85.6%)**	99 (11.4%)	22 (2.5%)	4 (0.5%)
Cordova	813	**730 (89.8%)**	69 (8.5%)	9 (1.1%)	5 (0.6%)
Drill	579	**515 (88.9%)**	63 (10.9%)	1 (0.2%)	0 (0.0%)
Flink	2,049	**1630 (79.6%)**	299 (14.6%)	96 (4.7%)	24 (1.2%)
Geode	150	**142 (94.7%)**	8 (5.3%)	0 (0.0%)	0 (0.0%)
Groovy	352	**325 (92.3%)**	25 (7.1%)	2 (0.6%)	0 (0.0%)
Guacamole	209	**197 (94.3%)**	8 (3.8%)	3 (1.4%)	1 (0.5%)
Hadoop	2,244	**2015 (89.8%)**	202 (9.0%)	21 (0.9%)	6 (0.3%)
HBase	1,142	**1028 (90.0%)**	101 (8.8%)	10 (0.9%)	3 (0.3%)
Hive	509	**472 (92.7%)**	34 (6.7%)	3 (0.6%)	0 (0.0%)
Hudi	184	**130 (70.7%)**	39 (21.2%)	13 (7.1%)	2 (1.1%)
Ignite	582	**505 (86.8%)**	65 (11.2%)	9 (1.5%)	3 (0.5%)
Impala	1,431	**1168 (81.6%)**	196 (13.7%)	45 (3.1%)	22 (1.5%)
Jackrabbit-Oak	168	**156 (92.9%)**	11 (6.5%)	1 (0.6%)	0 (0.0%)
Kafka	617	**545 (88.3%)**	60 (9.7%)	12 (1.9%)	0 (0.0%)
Lucene	201	**191 (95.0%)**	8 (4.0%)	2 (1.0%)	0 (0.0%)
Mesos	427	**364 (85.2%)**	46 (10.8%)	15 (3.5%)	2 (0.5%)
NetBeans	314	**259 (82.5%)**	48 (15.3%)	6 (1.9%)	1 (0.3%)
NiFi	221	**205 (92.8%)**	15 (6.8%)	0 (0.0%)	1 (0.5%)
OFBiz	215	**196 (91.2%)**	15 (7.0%)	4 (1.9%)	0 (0.0%)
Ozone	194	**180 (92.8%)**	13 (6.7%)	1 (0.5%)	0 (0.0%)
Qpid	214	**199 (93.0%)**	14 (6.5%)	1 (0.5%)	0 (0.0%)
Solr	377	**347 (92.0%)**	25 (6.6%)	4 (1.1%)	1 (0.3%)
Spark	1,773	**1556 (87.8%)**	185 (10.4%)	26 (1.5%)	6 (0.3%)
Thrift	183	**167 (91.3%)**	14 (7.7%)	1 (0.5%)	1 (0.5%)
Traffic Server	165	**152 (92.1%)**	12 (7.3%)	1 (0.6%)	0 (0.0%)
Wicket	212	**198 (93.4%)**	12 (5.7%)	2 (0.9%)	0 (0.0%)
All projects	18,440	**16,216 (87.9%)**	1803 (9.8%)	332 (1.8%)	89 (0.5%)

Table 7
Distribution of bugs over priority change patterns (RQ3).

Project	P1I_I	P1D_D	P2I_II	P2I_ID	P2I_DI	P2D_DD	P2D_ID	P2D_DI	P2R_ID	P2R_DI	PmI_I	PmI_ID	PmI_DI	PmI_IDI	PmI_DID	PmD_D	PmD_ID	PmD_DI	PmD_IDI	PmD_DID	PmR_ID	PmR_DI	PmR_IDI	PmR_DID	Number of patterns
ActiveMQ	71 32.9%	134 62.0%	1 0.5%			1 0.5%	1 0.5%	1 0.5%	5 2.3%	1 0.5%	1 0.5%	1 0.5%													**9**
Ambari	248 80.8%	51 16.6%	2 0.7%						3 1.0%	1 0.3%	1 0.3%					1 0.3%									**7**
Arrow	141 48.6%	127 43.8%	2 0.7%	1 0.3%		1 0.3%	1 0.3%	1 0.3%	11 3.8%	2 0.7%	1 0.3%	1 0.3%									1 0.3%				**12**
Axis2	248 58.4%	112 26.4%	4 0.9%	25 5.9%		3 0.7%	3 0.7%		16 3.8%	5 1.2%	1 0.2%	1 0.2%	1 0.2%			1 0.2%	2 0.5%	1 0.2%			2 0.5%				**15**
Camel	41 5.1%	726 90.0%				12 1.5%	2 0.2%		7 0.9%	6 0.7%	2 0.2%					7 0.9%	4 0.5%								**9**
CloudStack	501 57.6%	244 28.0%	18 2.1%	9 1.0%	1 0.1%	8 0.9%	7 0.8%	5 0.6%	35 4.0%	16 1.8%	6 0.7%	2 0.2%	1 0.1%			8 0.9%	2 0.2%	3 0.3%			2 0.2%	1 0.1%		1 0.1%	**19**
Cordova	160 19.7%	570 70.1%	4 0.5%	3 0.4%	4 0.5%	23 2.8%	2 0.2%	3 0.4%	19 2.3%	10 1.2%	1 0.1%		1 0.1%			2 0.2%	3 0.4%	1 0.1%			2 0.2%	2 0.2%	1 0.1%	1 0.1%	**19**
Drill	293 50.6%	222 38.3%	4 0.7%	2 0.3%	3 0.5%		10 1.7%		35 6.0%	9 1.6%	1 0.2%														**9**
Flink	1051 51.3%	579 28.3%	73 3.6%	30 1.5%	12 0.6%	40 2.0%	6 0.3%	13 0.6%	82 4.0%	43 2.1%	22 1.1%	22 1.1%	11 0.5%	1 0.1%	1 0.1%	17 0.8%	26 1.3%	3 0.1%			6 0.3%	4 0.2%	3 0.1%	4 0.2%	**22**
Geode	27 18.0%	115 76.7%	1 0.7%				1 0.7%		1 0.7%	5 3.3%															**6**
Groovy	166 47.2%	159 45.2%	1 0.3%	3 0.9%		1 0.3%	4 1.1%		14 4.0%	2 0.6%	1 0.3%							1 0.3%							**10**
Guacamole	11 5.3%	186 89.0%				4 1.9%			1 0.5%	3 1.4%						2 1.0%	2 1.0%								**7**
Hadoop	1322 58.9%	693 30.9%	38 1.7%	20 0.9%	6 0.3%	10 0.4%	20 0.9%	7 0.3%	84 3.7%	17 0.8%	7 0.3%	2 0.1%	2 0.1%			2 0.1%					12 0.5%		1 0.1%	1 0.1%	**17**
HBase	675 59.1%	353 30.9%	30 2.6%	7 0.6%	2 0.2%	3 0.3%	13 1.1%	3 0.3%	33 2.9%	10 0.9%	3 0.3%	3 0.3%				2 0.2%	2 0.2%				3 0.3%				**15**
Hive	366 71.9%	106 20.8%	6 1.2%	7 1.4%		4 0.8%	3 0.6%		12 2.4%	2 0.4%	1 0.2%	1 0.2%	1 0.2%												**11**
Hudi	104 56.5%	26 14.1%	7 3.8%	11 6.0%	2 1.1%	1 0.5%	1 0.5%		7 3.8%	10 5.4%	5 2.7%	1 0.5%	2 1.1%			2 1.1%	3 1.6%				1 0.5%	1 0.5%			**16**
Ignite	290 49.8%	215 36.9%	9 1.5%	11 1.9%	1 0.2%	9 1.5%	6 1.0%		24 4.1%	5 0.9%		2 0.3%	1 0.2%			1 0.2%	4 0.7%				2 0.3%		2 0.3%		**15**
Impala	638 44.6%	530 37.0%	43 3.0%	27 1.9%	8 0.6%	26 1.8%	6 0.4%	3 0.2%	51 3.6%	32 2.2%	13 0.9%	10 0.7%	3 0.2%		1 0.1%	6 0.4%	16 1.1%	3 0.2%	1 0.1%		9 0.6%	1 0.1%	2 0.1%	2 0.1%	**22**
Jackrabbit-Oak	76 45.2%	80 47.6%	2 1.2%			1 0.6%	1 0.6%		6 3.6%	1 0.6%													1 0.6%		**8**
Kafka	429 69.5%	116 18.8%	15 2.4%	10 1.6%	1 0.2%		1 0.2%	2 0.3%	23 3.7%	8 1.3%	4 0.6%	3 0.5%				2 0.3%	1 0.2%				2 0.3%				**14**
Lucene	62 30.2%	129 64.2%		2 1.0%					5 2.5%	1 0.5%	1 0.5%	1 0.5%													**7**
Mesos	246 57.6%	118 27.6%	6 1.4%	12 2.8%	1 0.2%	3 0.7%	1 0.2%	1 0.2%	18 4.2%	4 0.9%	4 0.9%	6 1.4%	1 0.2%			1 0.2%	1 0.2%				4 0.9%				**16**
NetBeans	107 34.1%	152 48.4%	9 2.9%	5 1.6%		6 1.9%	10 3.2%	2 0.6%	15 4.8%	1 0.3%	1 0.3%	1 0.3%	1 0.3%			1 0.3%		2 0.6%		1 0.3%					**15**
NiFi	111 50.2%	94 42.5%	1 0.5%	2 0.9%		1 0.5%	1 0.5%		8 3.6%	2 0.9%								1 0.5%							**9**
OFBiz	43 20.0%	153 71.2%	1 0.5%			1 0.5%	3 1.4%		7 3.3%	3 1.4%								4 1.9%							**8**
Ozone	156 80.4%	24 12.4%	5 2.6%	2 1.0%			1 0.5%		5 2.6%									1 0.5%							**7**
Qpid	50 23.4%	149 69.6%	1 0.5%	2 0.9%	1 0.5%		1 0.5%		6 2.8%	3 1.4%							1 0.5%								**9**
Solr	183 48.5%	164 43.5%	3 0.8%	1 0.3%			7 1.9%		11 2.9%	3 0.8%	1 0.3%	2 0.5%									1 0.3%		1 0.3%		**11**

(*continued on next page*)

Table 7 (*continued*).

Project	P1I_I	P1D_D	P2I_II	P2I_ID	P2I_DI	P2D_DD	P2D_ID	P2D_DI	P2R_ID	P2R_DI	PmI_I	PmI_ID	PmI_DI	PmI_IDI	PmI_DID	PmD_D	PmD_ID	PmD_DI	PmD_IDI	PmD_DID	PmR_ID	PmR_DI	PmR_IDI	PmR_DID	Number of patterns
Spark	662 37.3%	894 50.4%	39 2.2%	15 0.8%	9 0.5%	16 0.9%	10 0.6%	12 0.7%	61 3.4%	23 1.3%	6 0.3%	7 0.4%	2 0.1%		1 0.1%	5 0.3%	2 0.1%	2 0.1%			5 0.3%	1 0.1%	1 0.1%		**20**
Thrift	54 29.5%	113 61.7%		2 1.1%		5 2.7%		2 1.1%	3 1.6%	2 1.1%		1 0.5%				1 0.5%									**9**
Traffic Server	110 66.7%	42 25.5%	3 1.8%			1 0.6%			8 4.8%												1 0.6%				**6**
Wicket	30 14.2%	168 79.2%			1 0.5%	1 0.5%	3 1.4%	3 1.4%	2 0.9%	2 0.9%		1 0.5%						1 0.5%							**10**
All projects	**8762** **47.5%**	**7544** **40.9%**	**328** **1.8%**	**209** **1.1%**	**52** **0.3%**	**181** **1.0%**	**125** **0.7%**	**57** **0.3%**	**618** **3.4%**	**232** **1.3%**	**83** **0.5%**	**68** **0.4%**	**27** **0.1%**	**1** **<0.1%**	**3** **<0.1%**	**61** **0.3%**	**69** **0.4%**	**23** **0.1%**	**1** **<0.1%**	**1** **<0.1%**	**53** **0.3%**	**10** **<0.1%**	**12** **<0.1%**	**9** **<0.1%**	**24**
Number of projects	**32**	**32**	**27**	**23**	**14**	**24**	**27**	**13**	**32**	**30**	**21**	**19**	**12**	**1**	**3**	**17**	**14**	**12**	**1**	**1**	**15**	**6**	**8**	**5**	

Table 8
Distribution of priority change probability of different priorities (RQ3).

Project	Blocker			Critical			Major			Minor			Trivial		
	#OR	#PC	%PC	#OR	#PC	%PC	#OR	#PC	%PC	#OR	#PC	%PC	#OR	#PC	%PC
ActiveMQ	191	43	**22.5%**	451	53	11.8%	3,835	116	3.0%	718	18	2.5%	94	1	1.1%
Ambari	1,613	25	1.5%	4,282	43	1.0%	11,686	220	1.9%	610	24	3.9%	87	5	**5.7%**
Arrow	369	75	**20.3%**	262	35	13.4%	4,309	181	4.2%	851	23	2.7%	101	2	2.0%
Axis2	632	135	**21.4%**	555	74	13.3%	3,260	261	8.0%	549	30	5.5%	61	3	4.9%
Camel	72	47	**65.3%**	241	140	58.1%	4,441	642	14.5%	1,774	33	1.9%	97	5	5.2%
CloudStack	1,327	186	**14.0%**	2,391	285	11.9%	4,432	509	11.5%	666	45	6.8%	72	2	2.8%
Cordova	483	181	**37.5%**	700	200	28.6%	6,426	448	7.0%	1,903	80	4.2%	184	7	3.8%
Drill	267	47	**17.6%**	609	81	13.3%	4,394	469	10.7%	776	44	5.7%	35	3	8.6%
Flink	1,559	294	18.9%	2,057	508	24.7%	7,424	1,513	20.4%	1,716	145	8.4%	169	169	**100.0%**
Geode	17	4	**23.5%**	38	7	18.4%	5,188	132	2.5%	289	12	4.2%	55	3	5.5%
Groovy	327	70	**21.4%**	498	80	16.1%	5,181	199	3.8%	1,214	30	2.5%	79	2	2.5%
Guacamole	70	11	15.7%	77	28	**36.4%**	489	173	35.4%	388	14	3.6%	58	0	0.0%
Hadoop	2,293	297	13.0%	2,132	343	**16.1%**	16,946	1,574	9.3%	4,112	234	5.7%	822	59	7.2%
HBase	815	104	**12.8%**	1,366	157	11.5%	8,297	848	10.2%	2,452	139	5.7%	569	24	4.2%
Hive	499	39	**7.8%**	1,113	76	6.8%	11,513	368	3.2%	1,450	61	4.2%	280	5	1.8%
Hudi	327	62	19.0%	143	39	**27.3%**	867	142	16.4%	95	15	15.8%	3	0	0.0%
Ignite	600	80	13.3%	963	150	**15.6%**	5,489	393	7.2%	771	47	6.1%	93	4	4.3%
Impala	1,659	267	16.1%	1,522	369	24.2%	3,906	1,053	**27.0%**	903	100	11.1%	40	3	7.5%
Jackrabbit-Oak	139	15	**10.8%**	150	11	7.3%	2,766	132	4.8%	798	21	2.6%	54	2	3.7%
Kafka	791	86	10.9%	758	89	**11.7%**	4,862	463	9.5%	979	57	5.8%	119	6	5.0%
Lucene	162	21	13.0%	116	20	**17.2%**	2,959	154	5.2%	1,029	15	1.5%	148	3	2.0%
Mesos	429	73	17.0%	360	82	**22.8%**	3,953	323	8.2%	612	30	4.9%	78	2	2.6%
Netbeans	262	105	**40.1%**	382	56	14.7%	3,319	152	4.6%	1,036	57	5.5%	118	7	5.9%
NiFi	268	39	**14.6%**	357	26	7.3%	2,820	135	4.8%	1,050	37	3.5%	171	2	1.2%
OFBiz	84	27	**32.1%**	183	58	31.7%	2,880	110	3.8%	1,455	33	2.3%	376	10	2.7%
Ozone	238	16	6.7%	184	17	**9.2%**	2,005	169	8.4%	210	7	3.3%	43	0	0.0%
Qpid	192	32	**16.7%**	218	20	9.2%	3,938	157	4.0%	868	19	2.2%	111	2	1.8%
Solr	346	69	**19.9%**	399	46	11.5%	4,937	240	4.9%	1,701	50	2.9%	239	8	3.3%
Spark	1,239	311	25.1%	1,400	373	**26.6%**	11,002	1,111	10.1%	3,288	209	6.4%	364	24	6.6%
Thrift	137	30	**21.9%**	214	46	21.5%	2,136	104	4.9%	617	20	3.2%	156	2	1.3%
Traffic Server	124	11	8.9%	127	15	**11.8%**	2,642	147	5.6%	270	6	2.2%	42	0	0.0%
Wicket	61	24	**39.3%**	192	70	36.5%	2,991	108	3.6%	1,009	22	2.2%	168	4	2.4%
All projects	17,592	2826	**16.1%**	24,440	3597	14.7%	161,293	12,746	7.9%	36,159	1677	4.6%	5, 086	369	7.3%

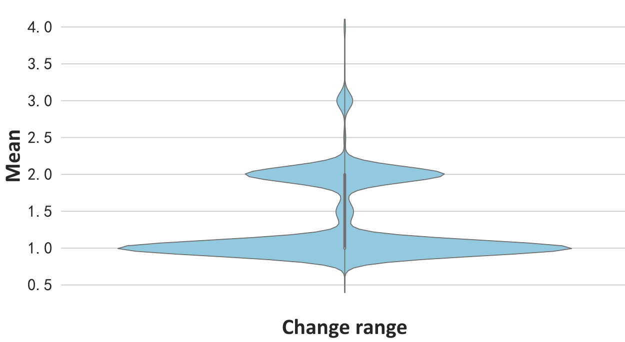

Fig. 7. Distribution of average bug change range (RQ3).

4.3.4. Relationship between bug priority change and priority itself

The probability of priority change of different priorities is listed in Table 8. In this table, #OR denotes the number of original priorities,[2] #PC denotes the number of priority changes. For 20 of the 32 projects, Blocker has the highest probability of priority change, and Critical has the highest probability of priority change in 10 of the remaining 12 projects. For all projects, as shown in Table 3, the priority change probability of Blocker is 16.1%, which is the highest of all priorities and much higher than 8.3% (i.e., the percentage of bugs with priority changes over all bugs of all the 32 projects). In addition, the reason why the priority change probability of Trivial in project *Flink* is 100% is that all the bug priority changes were made in a batch priority modification (i.e., the priorities of all bugs with priority changes were modified by someone at a time point). After removing the changes in *Flink*'s Trivial, we conducted a Spearman correlation coefficient analysis on the priority and its probability of change. The *p-value* is less than 0.05 and the correlation coefficient is 0.99, indicating a significant strong correlation between bug priority and the probability of priority change, i.e., the higher the bug priority, the greater the probability of bug priority change.

Table 9
Distribution of pairwise changes with different priorities (RQ3).

	Blocker	Critical	Major	Minor	Trivial	All
Blocker	0	1019	1480	298	29	2,826
Critical	1419	0	1789	370	19	3,597
Major	3755	4266	0	4383	342	12,746
Minor	209	312	988	0	168	1,677
Trivial	15	15	72	267	0	369
All	5398	5612	4329	5318	558	21,215

Table 9 lists the number of pairwise changes between different priorities of all projects. The left is the original priority, and the top is the changed priority. As we can see, most priorities are changed to the adjacent priorities; Major has the highest original priority, accounting for 60.1% (12,746 out of 21,215); except for Trivial, the numbers of changed priorities of all projects distribute over Blocker (5398), Critical (5612), Major (4329), and Minor (5318) in a balanced way in the sense that there is no big difference between the numbers.

[2] Note that the statistical unit here is a change record but not a bug, because a bug may have multiple priority changes.

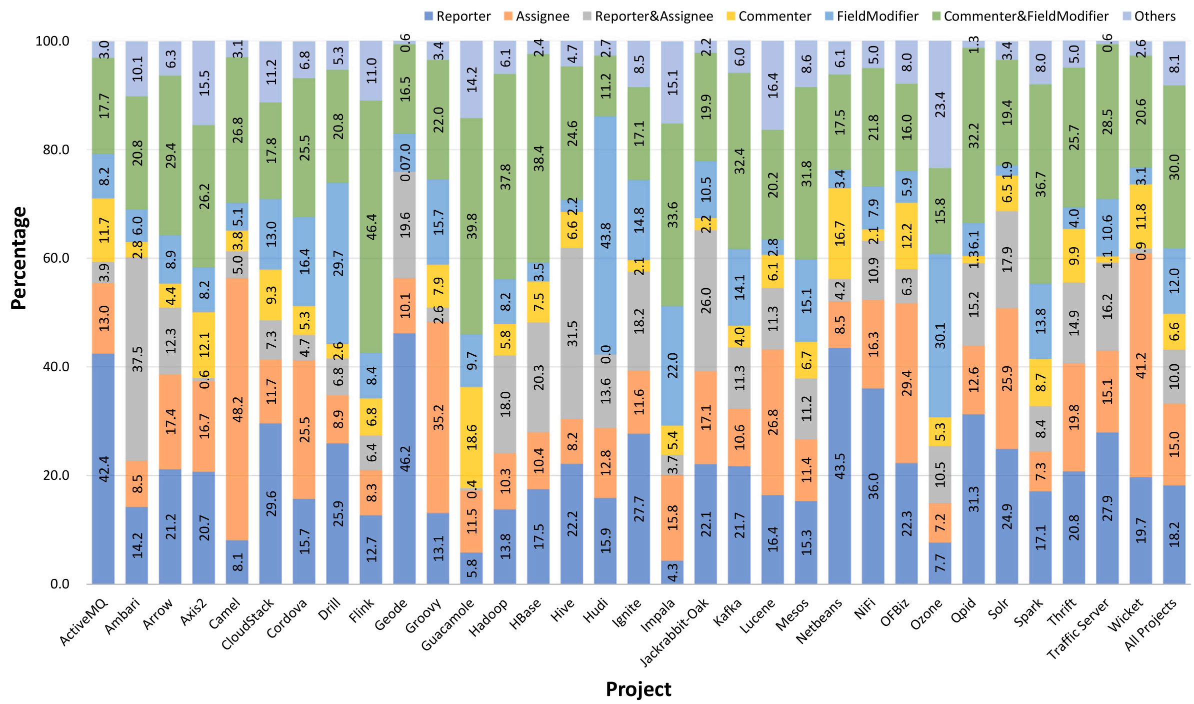

Fig. 8. Distribution of different types of PriorityModifier (RQ5).

4.4. Complexity of bugs (RQ4)

In this subsection, we calculate change complexity of bug-fixing commits and communication complexity of bugs with/without priority changes, in order to explore whether there is a significant difference on complexity between bugs with priority changes and bugs without priority changes. The bug-fixing commits of bugs and comments of bugs can respectively reflect the complexity of bugs from the perspective of code change and bug participants.

4.4.1. Change complexity of the bug-fixing commits for bugs

Table 10 shows the results of the Mann–Whitney U tests on the change complexity of bug-fixing commits for bugs with priority changes and bugs without priority changes. In this table, AveC denotes the average value of the corresponding change complexity metric of bug-fixing commits for bugs with priority changes and AveN denotes that of bugs without priority changes. *P-value* followed by $\star$ denotes that AveC is significantly less than AveN. As shown in Table 10, for change complexity metrics LOCM, NOFM, NOPM, and Entropy, the AveC of 29, 21, 24, and 25 out of 32 projects respectively are significantly greater than the corresponding AveN.

4.4.2. Communication complexity of bugs

Table 11 shows the results of the Mann–Whitney U tests on the communication complexity of bugs with priority changes and bugs without priority changes. We explore the communication complexity from the number of comments, the length of comments (the unit of length is the number of bytes), and the number of commenters. As shown in Table 11, for NOC, the AveC is significantly larger than the AveN in all the projects; for TLC and NOCR, the AveC of 29 and 31 of 32 projects respectively are significantly greater than the AveN.

4.5. Influence of human factors on bug priority changes (RQ5)

4.5.1. Proportion and distribution of different types of priority modifiers

The proportions of the five types of PriorityModifier are shown in Table 12, where # and % represent the number and percentage, respectively, of PriorityModifier of each type in the corresponding row. Taking all projects as a whole, 28.2% of the PriorityModifiers who have modified the bug priority are Reporter, 25.0% are Assignee, 71.3% have commented on the bugs, 81.7% have modified fields of the bug reports, and only 8.1% have not performed such activities. Please note that for each row in Table 12, the sum of the percentages is more than 100% since every PriorityModifier can belong to multiple types. To accurately describe the distribution of each PriorityModifier type, we further demonstrate the distribution in Fig. 8, in which the overlaps between different PriorityModifier types are explicitly shown. As we can see from Fig. 8, the distribution of different types of PriorityModifier in each project varies greatly. For all projects, among the PriorityModifiers who modify the bug priority, 18.2% are Reporter, 15.0% are Assignee, 10.0% are both Reporter and Assignee, 6.6% have only commented on the corresponding bugs, 12.0% have only modified fields of the bugs, 30% have both commented on and modified fields of the bugs, and only 8.1% have no such activities.

4.5.2. Distribution of bug participants prone to priority changes

The numbers of PRs, PAs, and PMs of each project are shown in Table 13. As we can see, there are 68 PRs in total, scattered in 15 projects, among which *Flink* has the largest number, 27 PRs. There are 13 PAs, which are only distributed in 5 projects, including 7 in *Impala*. There are even less PMs, with only 5 distributed in 5 projects.

5. Discussion

In this section, we interpret the study results according to the RQs and discuss the implications of the results for both practitioners and researchers.

5.1. Interpretation of study results

RQ1: Only a small proportion of bugs experience priority change, as the Apache projects have very clear standards for defining bug priorities (Apache, 2023), and participants are mostly able to correctly evaluate bug priorities based on existing documentation. For some projects, however, the probability of their bugs undergoing priority changes is relatively high compared to the average of all projects, which

Table 10
Average change complexity of bug-fixing commits for bugs in the selected projects (RQ4).

Project	LOCM				NOFM				NOPM				Entropy			
	AveC	AveN	*p-value*	*Effect size*	AveC	AveN	*p-value*	*Effect size*	AveC	AveN	*p-value*	*Effect size*	AveC	AveN	*p-value*	*Effect size*
ActiveMQ	189	162	<0.001	0.05	3.21	3.57	<0.001⋆	0.08	2.25	2.42	<0.001⋆	0.07	0.60	0.62	<0.001⋆	0.17
Ambari	142	143	<0.001⋆	0.02	4.13	4.09	<0.001	0.02	2.87	2.86	<0.001	<0.01	0.63	0.59	<0.001	0.01
Arrow	189	110	<0.001	0.22	3.65	3.52	<0.001	0.07	1.87	1.84	<0.001	0.02	0.68	0.61	<0.001	0.09
Axis2	212	148	<0.001	0.19	3.44	3.48	<0.001⋆	0.01	2.50	2.32	<0.001	0.08	0.53	0.47	<0.001	0.19
Camel	120	106	<0.001	0.07	3.36	3.55	<0.001⋆	0.05	2.38	2.39	<0.001⋆	0.01	0.70	0.65	<0.001	0.11
CloudStack	99	94	<0.001	<0.01	2.95	2.89	<0.001	0.01	2.18	2.19	<0.001⋆	<0.01	0.39	0.35	<0.001	0.14
Cordova	54	114	<0.001⋆	0.07	1.95	2.56	<0.001⋆	0.11	1.46	1.60	<0.001⋆	0.05	0.38	0.40	<0.001⋆	<0.01
Drill	308	201	<0.001	0.14	5.87	5.18	<0.001	0.08	3.76	3.28	<0.001	0.09	0.73	0.66	<0.001	0.05
Flink	213	181	<0.001	0.08	5.11	4.65	<0.001	0.08	3.32	3.10	<0.001	0.07	0.65	0.66	<0.001⋆	0.02
Geode	367	271	<0.001	0.16	4.00	4.95	<0.001⋆	0.13	2.80	3.08	<0.001⋆	0.09	0.62	0.65	<0.001⋆	0.08
Groovy	102	101	<0.001	0.02	2.80	2.69	<0.001	0.05	2.27	2.21	<0.001	0.04	0.61	0.65	<0.001⋆	0.03
Guacamole	139	140	<0.001⋆	0.06	3.50	3.00	<0.001	0.14	1.74	1.88	<0.001⋆	0.02	0.43	0.39	<0.001	0.01
Hadoop	112	94	<0.001	0.05	3.83	3.01	<0.001	0.13	2.82	2.24	<0.001	0.18	0.64	0.55	<0.001	0.16
HBase	178	126	<0.001	0.14	4.20	3.44	<0.001	0.15	2.59	2.24	<0.001	0.16	0.57	0.48	<0.001	0.13
Hive	178	144	<0.001	0.07	4.13	3.48	<0.001	0.12	2.63	2.30	<0.001	0.14	0.55	0.47	<0.001	0.21
Hudi	147	120	<0.001	0.14	5.57	4.00	<0.001	0.34	4.19	3.11	<0.001	0.38	0.74	0.63	<0.001	0.36
Ignite	251	210	<0.001	0.19	6.88	5.24	<0.001	0.16	4.42	3.46	<0.001	0.20	0.78	0.64	<0.001	0.21
Impala	105	80	<0.001	0.10	3.63	2.94	<0.001	0.17	2.16	1.98	<0.001	0.14	0.56	0.52	<0.001	0.09
Jackrabbit-Oak	154	138	<0.001	0.02	3.71	3.60	<0.001	0.02	2.56	2.30	<0.001	0.09	0.62	0.60	<0.001	0.05
Kafka	248	161	<0.001	0.21	6.41	4.66	<0.001	0.22	3.76	2.94	<0.001	0.27	0.80	0.69	<0.001	0.27
Lucene	154	146	<0.001	0.08	4.03	4.17	<0.001⋆	0.14	2.68	2.62	<0.001	0.21	0.71	0.64	<0.001	0.15
Mesos	121	60	<0.001	0.36	1.30	1.90	<0.001⋆	0.25	1.20	1.34	<0.001⋆	0.20	0.17	0.29	<0.001⋆	0.48
NetBeans	217	145	<0.001	0.13	3.28	3.43	<0.001⋆	0.10	2.39	2.51	<0.001⋆	0.03	0.43	0.58	<0.001⋆	0.14
NiFi	209	136	<0.001	0.10	4.26	3.47	<0.001	0.15	2.96	2.45	<0.001	0.17	0.61	0.59	<0.001	0.20
OFBiz	38	36	<0.001	<0.01	2.43	1.60	<0.001	0.18	2.00	1.37	<0.001	0.26	0.25	0.20	<0.001	0.05
Ozone	199	149	<0.001	0.16	5.67	4.68	<0.001	0.17	4.01	3.21	<0.001	0.23	0.67	0.56	<0.001	0.37
Qpid	154	147	<0.001	0.01	3.98	4.37	<0.001⋆	0.07	2.66	2.51	<0.001	0.05	0.58	0.57	<0.001	0.07
Solr	161	148	<0.001	0.13	4.13	4.21	<0.001⋆	0.15	2.71	2.62	<0.001	0.22	0.73	0.63	<0.001	0.14
Spark	94	74	<0.001	0.10	3.21	3.16	<0.001	0.02	2.47	2.46	<0.001	0.02	0.60	0.63	<0.001	0.06
Thrift	125	58	<0.001	0.30	3.16	1.95	<0.001	0.39	1.96	1.44	<0.001	0.47	0.46	0.31	<0.001	0.23
Traffic Server	100	99	<0.001	0.01	3.23	3.47	<0.001⋆	0.01	1.96	1.93	<0.001	0.02	0.43	0.41	<0.001	0.08
Wicket	114	90	<0.001	0.08	3.21	2.54	<0.001	0.21	1.99	1.93	<0.001	0.09	0.48	0.46	<0.001	0.09

could be related to the way of bug management. For example, *Flink* has a bot, i.e., Apache Flink Jira Bot, which assists developers in the management of issues. For some other projects, the reason for the high probability of bug priority change may be that the project size is small (e.g., *Guacamole*) or there are batch priority modifications (e.g., *Flink* and *Impala*).

RQ2: The minimum time interval for each project is only 9 s to 346 s. Due to the fact that the PriorityModifier and bug reporter may be different participants, we only filtered out priority modifications made by the same PriorityModifier within 5 min after the bug is reported, while retaining priority modifications made by different PriorityModifiers. The median priority change time interval of bugs is rather short for most projects, while the average value is much longer than the median. This is because in each project there are some bugs with a very long priority change time interval. The relatively small median priority change time interval of the bugs in 28 out of 32 projects indicates that at least a half of the bugs in those projects were reconsidered in terms of bug priority in a few days. The time interval presents this distribution because priority is usually either modified shortly after a bug is reported, or no one claims the bug after it is reported, resulting in the bug to be processed after a long period of time, and the priority changes accordingly.

The priority changes of most bugs in 24 out of 32 projects happened BEFORE. To understand the reasons behind this, we further checked the priority changes that occurred during BEFORE, and the vast majority of the changes occurred in the situation of S4.2 in Fig. 1 (i.e., the priority changes occurred before the bug was linked to other bugs/issues). Therefore, this to some extent explains the reason for the priority change (i.e., the bug is linked to other bugs), which is consistent with the research results of Almhana et al. (2020). The reason why the change phase for a small number of bugs is REOPEN is because some bugs start to be fixed after being reopened, and priority changes may also occur. This is a normal phenomenon. However, some bugs still have priority changes after being resolved or closed (i.e., AFTER). Further exploration is needed to uncover the reasons for this phenomenon.

RQ3: Among the bugs that have priority changes, most bugs have only one priority change and usually change to adjacent priorities, which resonates with the priority change range of 1 for most bugs. This happens because priority is used to describe the fixing schedule for bugs, and the fixing content represented by the bug itself does not change after it is reported. The fixing work is related to the fixing content, and consequently the times and range of priority changes are relatively small. Accordingly, in the priority change patterns, most bug priority change patterns focus on P1I_I and P1D_D. In addition, P2R_ID and P2R_DI have the largest number of bugs. This is because among the bugs whose priority is changed twice, the number of bugs that the priority restores finally is the largest. This back and forth modification of the priority is because some PriorityModifier is uncertain about the priority.

In addition, we are also curious about why some bugs have undergone so many priority changes, and we manually checked the history and comments of bugs whose priorities have changed more than three times. We used open coding and constant comparison of Grounded Theory (Stol et al., 2016) to generate concepts and categories of reasons for that bugs underwent priority changes more than three times. Grounded Theory (GT) is a bottom-up approach that focuses on theory generation rather than extending or validating existing theory. Open coding and constant comparison are two techniques widely used in Grounded Theory for qualitative data analysis.

Table 11
Average communication complexity for bugs in the selected projects (RQ4).

Project	NOC				TLC				NOCR			
	AveC	AveN	*p-value*	*Effect size*	AveC	AveN	*p-value*	*Effect size*	AveC	AveN	*p-value*	*Effect size*
ActiveMQ	6.10	3.47	<0.001	0.61	2,632	1482	<0.001	0.19	2.93	1.96	<0.001	0.68
Ambari	4.87	3.38	<0.001	0.57	2,581	1683	<0.001	0.12	2.87	2.27	<0.001	0.56
Arrow	5.34	3.12	<0.001	0.51	2,696	1676	<0.001	0.08	2.40	1.72	<0.001	0.67
Axis2	5.18	2.86	<0.001	0.73	2,261	1069	<0.001	0.41	2.75	1.88	<0.001	0.62
Camel	4.72	3.26	<0.001	0.40	1,858	1216	<0.001	0.15	2.03	1.69	<0.001	0.37
CloudStack	8.69	5.80	<0.001	0.25	5,401	4488	<0.001	0.05	3.40	2.14	<0.001	0.93
Cordova	6.63	4.56	<0.001	0.31	4,067	3027	<0.001	0.02	2.83	2.10	<0.001	0.48
Drill	5.61	4.49	<0.001	0.16	4,716	6714	<0.001⋆	0.01	2.50	1.97	<0.001	0.45
Flink	8.94	5.06	<0.001	0.52	5,881	3512	<0.001	0.09	3.22	2.05	<0.001	0.85
Geode	5.63	4.49	0.005	0.16	7,425	8363	0.336⋆	0.01	2.36	1.78	<0.001	0.49
Groovy	5.39	3.24	<0.001	0.56	2,241	1148	<0.001	0.36	2.48	1.83	<0.001	0.57
Guacamole	5.24	2.85	<0.001	0.52	4,101	1533	<0.001	0.27	2.23	1.59	<0.001	0.58
Hadoop	15.8	9.43	<0.001	0.60	11,643	7169	<0.001	0.30	4.92	3.53	<0.001	0.66
HBase	21.22	11.00	<0.001	0.74	16,647	7587	<0.001	0.35	5.11	3.78	<0.001	0.62
Hive	10.49	6.75	<0.001	0.49	11,904	9749	<0.001	0.03	3.73	2.82	<0.001	0.53
Hudi	2.99	2.94	0.005	0.01	2,933	4487	0.030⋆	0.07	1.64	1.28	<0.001	0.45
Ignite	4.92	3.50	<0.001	0.34	2,979	1466	<0.001	0.23	2.52	2.10	<0.001	0.31
Impala	5.49	2.91	<0.001	0.66	3,418	1888	<0.001	0.23	2.76	1.84	<0.001	0.71
Jackrabbit-Oak	7.31	4.47	<0.001	0.59	2,244	1864	<0.001	0.02	2.79	2.08	<0.001	0.68
Kafka	9.34	4.39	<0.001	0.78	7,424	3672	<0.001	0.15	3.86	2.32	<0.001	0.77
Lucene	11.49	7.40	<0.001	0.47	11,498	2992	<0.001	0.37	3.72	2.84	<0.001	0.52
Mesos	5.86	3.14	<0.001	0.72	10,100	2622	<0.001	0.23	2.94	1.85	<0.001	0.84
NetBeans	6.39	2.59	<0.001	0.31	7,245	6746	<0.001	<0.01	2.66	1.47	<0.001	1.10
NiFi	6.88	4.52	<0.001	0.40	4,064	2702	<0.001	0.15	2.72	2.07	<0.001	0.52
OFBiz	7.67	4.39	<0.001	0.60	3,601	1481	<0.001	0.38	2.96	2.19	<0.001	0.65
Ozone	5.04	3.24	<0.001	0.38	5,289	3524	<0.001	0.17	2.81	2.07	<0.001	0.48
Qpid	4.97	2.93	<0.001	0.72	2,012	926	<0.001	0.33	2.50	1.90	<0.001	0.59
Solr	12.76	6.40	<0.001	0.79	5,575	2738	<0.001	0.38	4.18	2.82	<0.001	0.72
Spark	5.93	3.41	<0.001	0.62	3,573	1297	<0.001	0.14	3.13	2.17	<0.001	0.63
Thrift	7.31	4.58	<0.001	0.48	2,848	1677	<0.001	0.23	2.73	2.41	0.055	0.22
Traffic Server	7.81	4.72	<0.001	0.40	3,897	2074	<0.001	0.08	3.19	2.16	<0.001	0.77
Wicket	6.32	3.91	<0.001	0.54	1,981	1176	<0.001	0.27	2.76	2.02	<0.001	0.62

The analysis process includes the following steps: (1) The second author used open coding to code the extracted data items (i.e., the comment content of the bug) of bugs with more than three priority changes to generate codes. When the content of the bug comments was unclear and the second author was confused when coding the extracted data, a physical meeting was arranged with the first author to resolve this confusion. (2) The second author applied constant comparison to compare codes identified in one piece of data with codes appearing in other data to identify codes with similar semantics. The second author continued to group similar codes into high-level concepts and categories, and the classification process was iterative, with the second author constantly switching back and forth between categories, content of comments, to modify and refine the categories. (3) Afterwards, the third and fourth authors checked and verified the results of the data analysis (i.e., codes, concepts, and categories). Disagreements were resolved through a meeting using a negotiated agreement approach (Campbell et al., 2013) to improve the reliability of data analysis results.

In Table 14, we summarize the reasons why those bugs underwent priority changes for more than 3 times. To illustrate each reason, we provide a description, an instance, and the number of corresponding bugs. From the distribution of bug numbers corresponding to each reason in Table 14, we can see that when a bug undergoes multiple priority changes, the likely reason is that the bug itself is relatively complex, which in turn leads to a more complex process of fixing the bug, increasing uncertainty and affecting the decision of PriorityModifier.

For the priority change trend of all bugs, increase and decrease account for the vast majority, with more increase than decrease of priority change. The priority change is also related to the priority itself; the higher the priority, the greater the probability of priority change. The reason for this phenomenon is that participants are usually more cautious about assigning a higher priority to a bug, as a higher priority usually has a greater impact on their work schedule and release planning (Apache, 2023). This leads to the situation that some of such bugs were underestimated with respect to their priority when a bug was reported, and even if a higher priority is assigned, participants are relatively more likely to change their priority.

RQ4: Surprisingly, for most projects, the average change complexity of bug-fixing commits for bugs with priority changes is significantly higher than that of bugs without priority changes. This means no matter the trend of priority changes of bugs, the bug-fixing commits for such bugs tend to be more complex than that for bugs without priority changes. In other words, bugs with higher change complexity of bug-fixing commits are more likely to undergo priority changes. We can imagine that it is more difficult to precisely estimate the consequence of a bug with more complex bug-fixing changes. The same is true for the communication complexity. The communication complexity of bugs with priority changes is significantly greater than that of bugs without priority changes. The complexity of comments shows the intensity of the bug discussion among bug participants. In other words, the more comments of bugs, the more likely they are to undergo priority changes. Thus, to some extent, the reason why bugs undergo priority changes is that the fixing of such bugs or bugs themselves is more complex than bugs without priority changes.

RQ5: The study shows that 43.2% of PriorityModifier are Reporter or Assignee, and most PriorityModifier have commented on the corresponding bug or modified other bug fields except for the Priority field. This is because PriorityModifier is usually familiar with the bug, and also reveals that priority changes may be the result of discussion among PriorityModifier. The proportion of PriorityModifier varies between different projects, as there are significant differences in the size and number of participants of these projects.

Table 12
Proportion of different types of PriorityModifier (RQ5).

Project	Reporter # (%)	Assignee # (%)	Commenter # (%)	FieldModifier # (%)	Others # (%)	Total
ActiveMQ	100 (46.3%)	39 (16.9%)	178 (77.1%)	180 (77.9%)	7 (3.0%)	231
Ambari	164 (51.7%)	146 (46.1%)	218 (68.8%)	270 (85.2%)	32 (10.1%)	317
Arrow	106 (33.5%)	94 (29.7%)	226 (71.5%)	270 (85.4%)	20 (6.3%)	316
Axis2	107 (21.3%)	87 (17.3%)	348 (69.2%)	330 (65.6%)	78 (15.5%)	503
Camel	113 (13.0%)	461 (53.2%)	738 (85.1%)	798 (92.0%)	27 (3.1%)	867
CloudStack	379 (36.9%)	195 (19.0%)	675 (65.7%)	784 (76.3%)	115 (11.2%)	1,027
Cordova	187 (20.4%)	277 (30.2%)	629 (68.7%)	760 (83.0%)	62 (6.8%)	916
Drill	211 (32.8%)	101 (15.7%)	355 (55.1%)	559 (86.8%)	34 (5.3%)	644
Flink	501 (19.1%)	387 (14.7%)	2042 (77.7%)	2096 (79.7%)	288 (11.0%)	2,629
Geode	104 (65.8%)	47 (29.7%)	72 (45.6%)	144 (91.1%)	1 (0.6%)	158
Groovy	60 (15.7%)	144 (37.8%)	275 (72.2%)	326 (85.6%)	13 (3.4%)	381
Guacamole	14 (6.2%)	27 (11.9%)	162 (71.7%)	145 (64.2%)	32 (14.2%)	226
Hadoop	798 (31.8%)	710 (28.3%)	2059 (82.1%)	2150 (85.8%)	152 (6.1%)	2,507
HBase	480 (37.7%)	390 (30.7%)	1168 (91.8%)	1129 (88.8%)	31 (2.4%)	1,272
Hive	295 (53.7%)	218 (39.7%)	452 (82.3%)	462 (84.2%)	26 (4.7%)	549
Hudi	76 (29.5%)	68 (26.4%)	75 (29.1%)	247 (95.7%)	7 (2.7%)	258
Ignite	310 (46.0%)	201 (29.8%)	367 (54.5%)	579 (85.9%)	57 (8.5%)	674
Impala	144 (8.0%)	350 (19.5%)	1074 (59.9%)	1409 (78.6%)	270 (15.1%)	1,792
Jackrabbit-Oak	87 (48.1%)	78 (43.1%)	148 (81.8%)	169 (93.4%)	4 (2.2%)	181
Kafka	231 (33.0%)	153 (21.8%)	478 (68.2%)	594 (84.7%)	42 (6.0%)	701
Lucene	59 (27.7%)	81 (38.0%)	162 (76.1%)	163 (76.5%)	35 (16.4%)	213
Mesos	135 (26.5%)	115 (22.5%)	353 (69.2%)	421 (82.5%)	44 (8.6%)	510
Netbeans	180 (47.7%)	48 (12.7%)	283 (75.1%)	232 (61.5%)	23 (6.1%)	377
NiFi	112 (46.9%)	65 (27.2%)	152 (63.6%)	202 (84.5%)	12 (5.0%)	239
OFBiz	68 (28.6%)	85 (35.7%)	190 (79.8%)	183 (76.9%)	19 (8.0%)	238
Ozone	38 (18.2%)	37 (17.7%)	77 (36.8%)	147 (70.3%)	49 (23.4%)	209
Qpid	107 (46.5%)	64 (27.8%)	191 (83.0%)	214 (93.0%)	3 (1.3%)	230
Solr	177 (42.9%)	181 (43.8%)	357 (86.4%)	355 (86.0%)	14 (3.4%)	413
Spark	517 (25.5%)	319 (15.7%)	1402 (69.1%)	1597 (78.7%)	168 (8.0%)	2,028
Thrift	72 (35.6%)	70 (34.7%)	153 (75.7%)	164 (81.2%)	10 (5.0%)	202
Traffic Server	79 (44.1%)	56 (31.3%)	131 (73.2%)	169 (94.4%)	1 (0.6%)	179
Wicket	47 (20.6%)	96 (42.1%)	199 (87.3%)	187 (82.0%)	6 (2.6%)	228
All projects	**5267 (28.2%)**	**4680 (25.0%)**	**13,330 (71.3%)**	**15,285 (81.7%)**	**1524 (8.1%)**	**18,708**

Finally, we find that the priority change does have the influence of human factors. However, these numbers are not large, and there are no such participants in many projects, which shows that the impact of human factors is limited. And this seems to be related to the scale of the project. The larger the project, the more participants there will be. The reason for the large number of PRs in project *Flink* is that Flink Jira Bot exists in the project, and there are also some batch modifications, which indicates that the priority change is also related to the development style of the project.

5.2. Implications for practitioners

The results of this study imply a number of points for practitioners, which are presented as follows.

- **This study helps practitioners have a better understanding of the probability of bug priority change for specific projects.** Due to the varying probability of bug priority change for each project, and even significant differences in some projects, practitioners can evaluate the probability of bug priority change for specific projects based on their characteristics (such as management style and project size), in order to better manage bugs.
- **Practitioners need to pay attention to the priority changes that occur at different stages of the bug life cycle.** This study shows that most bug priority changes occur shortly after a bug is reported and before it begins to be processed. Therefore, practitioners can pay more attention to bug priority changes in the early stages of bug fixing to arrange their work schedule reasonably. After the bug is resolved or closed, practitioners should be cautious of changing its priority. Even though the priority change on the resolved or closed bug would not impact the bug fixing process, it may influence the future tasks based on bug priority, such as bug priority prediction and workload estimation.
- **Bug reports need to display the process of priority changes.** Compared to displaying only the current priority, bug reports showing the process of bug priority changes (such as priority change patterns) can help practitioners understand and fix bugs more efficiently.
- **Bugs that require relatively more complex bug-fixing commits or are discussed more during bug fixing tend to undergo priority changes.** This is evidenced by the fact that in most projects, bugs with priority changes have significantly higher code change complexity and communication complexity in bug fixing than bugs without priority changes, as shown in Tables 10 and 11, respectively.
- **Pay attention to the needs of bug reporters and assignees.** We noticed that a considerable portion of the bug priority changes are modified by the bug reporter and the assignee, indicating that bug reporters and assignees play important roles in the bug fixing process, and understanding the needs of reporters and assignees is crucial for optimizing workflow. Practitioners can understand the needs and expectations of these two roles through regular user feedback and surveys, in order to provide better support and services.
- **There is a need to standardize the priority allocation process in order to reduce the impact of human factors during priority allocation.** In some projects, priorities of bugs reported by a few participants or priorities allocated by a small number of participants are more likely to be modified, and a few participants tend to modify priorities. Following a standardized priority allocation process can help to alleviate the impact of potential biases and preferences of such participants in bug priority management.

Table 13
Distribution of PRs, PAs, and PMs in selected projects (RQ5).

Project	#PR	#PA	#PM
ActiveMQ	0	0	0
Ambari	0	0	0
Arrow	0	0	0
Axis2	2	0	0
Camel	1	0	0
CloudStack	5	2	0
Cordova	1	0	1
Drill	3	0	0
Flink	27	2	1
Geode	0	0	0
Groovy	0	0	0
Guacamole	0	0	1
Hadoop	8	0	0
HBase	1	0	0
Hive	0	0	0
Hudi	3	1	0
Ignite	0	0	0
Impala	6	7	1
Jackrabbit-Oak	0	0	0
Kafka	2	0	0
Lucene	0	0	1
Mesos	2	1	0
NetBeans	0	0	0
NiFi	0	0	0
OFBiz	0	0	0
Ozone	1	0	0
Qpid	1	0	0
Solr	0	0	0
Spark	5	0	0
Thrift	0	0	0
Traffic Server	0	0	0
Wicket	0	0	0
All projects	68	13	5

5.3. Implications for researchers

The results of this study also imply a number of points for researchers, which are presented as follows.

- **This study identifies possible dimensions where researchers can explore the reasons for relatively high priority change probabilities of certain projects.** Since bugs with priority changes have higher change complexity of bug-fixing commits, it is necessary to make clear the reasons for why bugs in specific projects have high priority change probabilities, in order to find a way to reduce maintenance cost. Our study also identifies several aspects to be further explored by researchers, such as the way of project management (e.g., Flink Jira Bot in *Flink*), project size.
- **The phenomenon that a small proportion of bugs occurred priority changes after being resolved or closed deserves an in-depth study.** Priority represents the urgency of a bug, and modifying the priority after the bug is resolved or closed does not affect the progress of bug fixing. Therefore, the reasons behind this unreasonable behavior are worth further investigation.
- **The bug priority change process may provide useful information to help researchers improve the bug priority prediction models or bug-fixing time prediction models.** Due to the lack of attention paid to bug priority changes, they have hardly been applied to the training of various prediction models. The 24 change patterns proposed in this study characterize the process of priority changes for each bug, which may help to improve the performance of the prediction models.
- **Further investigation on the reasons for bug priority changes is needed.** Currently, the reasons have merely been investigated from the perspective of the change complexity of bug-fixing commits and the communication complexity of bugs. Other perspectives, such as project-specific factors, should also be further studied. To estimate bug priorities more accurately thereby optimizing release planning and task assignment in project management, it is worthwhile to look into diverse factors that drive bug priority changes in depth.
- **It is worthwhile to conduct in-depth research on the role of the main participants in bug fixing.** Our research indicates that a significant portion of PriorityModifiers are bug reporters and assignees, and researchers can gain a deeper understanding of their specific roles in bug discovery, resolution, and validation to better understand the dynamics of bug fixing.

6. Threats to validity

There are several threats to the validity of the study results. We discuss these threats according to the guidelines in Runeson and Höst (2009). Please note that internal validity is not discussed since we did not study causal relationships.

6.1. Construct validity

Construct validity is concerned with whether the values of the variables (listed in Table 1) we obtained are in line with the real values that we expected. A potential threat to construct validity is that not all bugs resolved are linked to the corresponding commits. Due to different developer habits and development cultures, committers may not explicitly mention the ID of the bug resolved in the corresponding commit message, which may negatively affect the representativeness of the collected bugs and further influence the accuracy of defect density and the time taken to resolve bugs. Through our analysis (the analysis results are not shown in this paper due to its deviation from the focus of this paper), we confirmed that the committers who explicitly mention the bug ID do not come from a small group of specific developers. Therefore, this threat is mitigated to some extent.

Another threat is that a bug will probably be reopened and the priority may be changed again, which results in the incompleteness of priority changes of the bug and thus affects the study results. We analyzed the data of the bugs that were resolved or closed before December 1, 2019 and were not reopened again before February 4, 2021. It means that the status of such bugs had been stable for one year and two months after they were resolved or closed. We believe that the likelihood of such bugs being reopened is significantly decreased. Therefore, this threat is alleviated to a large extent.

6.2. External validity

External validity is concerned with the generalizability of the study results. First, we only consider the five bug priorities of Blocker, Critical, Major, Minor, and Trivial in JIRA. The research results may not be generalized to other priorities in JIRA or priorities in other issue tracking systems. Then another potential threat to external validity is whether the selected projects are representative enough. As presented in Section 3.3, we applied a set of criteria to select projects. We tried to include as many Apache projects that meet the selection criteria as possible. Furthermore, the selected projects cover different languages and different application domains, and differ in code repository size and development duration. This indicates an improved representativeness of the selected projects. Finally, since only OSS projects were selected, the study results and findings may not be generalized to closed source software projects.

6.3. Reliability

Reliability refers to whether the study yields the same results when replicated by other researchers. A potential threat is related to the

Table 14
Reasons for bug priority change of more than 3 times.

#	Reason	Description	Instance	#Bug
RS1	PriorityModifiers fail to reach agreement with the bug priority.	This can be embodied in that two or more modifiers change the priority back and forth in the history, or some commenters raise objections to the priority allocation in the comments.	*CLOUDSTACK-1673*, the priority sequence of the bug is 23212. A comment mentioned "When you mark bug as blocker, please make sure to mention why its blocker."	26
RS2	PriorityModifiers are uncertain about the bug priority.	One or more PriorityModifiers change the priority repeatedly, or one or more PriorityModifiers change the priority and then change it back to the original priority.	*IMPALA-1755*, the priority sequence of the bug is 32121. A comment mentioned "I think it was marked as a blocker since it was a regression, but given that no one seemed to notice I would say downgrade the priority."	14
RS3	The bug priority is version related.	For example, a bug is Blocker in the current release but not Blocker in the next release, or it exists in the current release but does not exist in the next release, or the bug priority may be lowered before the next release.	*AXIS2-653*, the priority sequence of the bug is 31312. A comment mentioned "Reducing the priority as: 1. the discussion on this is yet to be finished; 2. not a blocker for the next release."	13
RS4	The bug is unreproducible.	After one participant reports the bug, other participants cannot reproduce the bug for some reasons, such as differences in the software environment.	*AXIS2-3099*, the priority sequence of the bug is 312124. A comment mentioned that "Not a blocker as we are not able to reproduce the issue."	6
RS5	The bug priority is changed according to the bug fixing progress.	The bug is too complex, or the participants do not fully understand the bug, resulting in repeated modification of the priority in the bug-fixing process.	*IMPALA-2982*, the priority sequence of the bug is 212134. Two comments mention that "Bringing this back to Blocker level as the excessive logging is a major supportability issue" and "Reduced priority as the estimated impact is not low."	4
RS6	The bug priority is changed by a Jira bot.	There is a bug management robot called Flink Jira Bot used in project *Flink*. It will automatically check the bug progress. If the bug is not updated for a long time, it will decrease the bug priority.	*FLINK-18574*, the priority sequence of the bug is 32342. 234 in 32342 is the priority change made by the robot.	14
RS7	The bug priority is changed for unknown reasons.	No explicit reasons can be found after checking the history and comments of the bug. This may be because the bug links to other bugs or another instance (including the bugs on JIRA and GitHub, or instance on Azure).	*FLINK-17260*, the priority sequence of the bug is 432132. In the comment, the participants associate it with another instance on Azure.	12

implementation of related software tools for data collection. The tools were mainly implemented by the third author, and the code of the key functionalities had been regularly reviewed by the first and second authors. Furthermore, sufficient tests were performed to ensure the correctness of the calculation of data items. Hence, this threat to reliability had been alleviated.

Another threat is related to the correctness of the Mann–Whitney U tests. Since we used IBM SPSS (a widely-used professional tool for statistics) and the Scipy library in Python (a scientific computing-related package of Python) to run the tests. Hence, we believe that the threat is minimized.

7. Conclusions and future work

In this work, we investigated the phenomenon of bug priority changes by analyzing bugs and related data collected from 32 non-trivial OSS projects of Apache Software Foundation. We identified 24 patterns characterizing the process of bug priority changes. The main findings are summarized as follows:

- A small proportion (only 8.3%) of bugs of the selected projects undergo priority changes.
- In 28 out of 32 projects, the median time interval of priority changes is less than ten days.
- In 24 out of 32 projects, the number of bugs whose priority changes happened before the bug-fixing process accounts for the largest.
- At least 81.6% of the bugs with priority changes of each project undergo only one priority change, and no more than 8.2% of the bugs with priority changes of each project undergo three or more priority changes.
- Each project covers 6 to 22 bug priority change patterns, none of the projects covers all the 24 patterns, and most of the bugs with priority changes increase the bug priority finally.
- Most priority changes tend to shift the priority to its adjacent priority and a higher priority holds a greater probability to undergo priority change.
- In most of the projects, bugs that undergo priority changes have significantly higher change complexity of bug-fixing commits and communication complexity than bugs that do not undergo priority changes.
- 43.2% of the participants who make priority changes are the reporter or assignee of the bug, 71.3% of them make comments on the bug, and 81.7% of them modify other fields of the bug except the priority field.
- In around a half of the 32 projects, bugs reported by a few participants are more likely to be modified; in 5 out of the 32 projects, the priorities allocated by a few participants are more likely to be modified; and in 5 out of the 32 projects, there is one participant who is prone to modifying the bug priority.

Based on the results of this study, our future research will focus on the following directions: first, to investigate bug priority changes in other OSS ecosystems using different issue tracking systems with distinct priority models; second, to study the relationship between priority changes and different issue types or project types; third, to explore how human factors specifically affect bug priority changes.

CRediT authorship contribution statement

Zengyang Li: Writing – original draft, Visualization, Resources, Methodology, Investigation, Funding acquisition, Data curation. **Guangzong Cai:** Writing – original draft, Visualization, Software, Investigation, Data curation. **Qinyi Yu:** Writing – original draft, Software, Investigation, Data curation. **Peng Liang:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Ran Mo:** Writing – review & editing, Methodology, Funding acquisition. **Hui Liu:** Writing – review & editing, Methodology, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the link to our dataset in the reference (Li et al., 2024).

Acknowledgments

This work is supported by the Natural Science Foundation of Hubei Province of China under Grant No. 2021CFB577, the National Natural Science Foundation of China under Grant Nos. 62176099 and 62172311, and the Knowledge Innovation Program of Wuhan-Shuguang Project under Grant No. 2022010801020280.

References

Akbarinasaji, S., Caglayan, B., Bener, A., 2018. Predicting bug-fixing time: A replication study using an open source software project. J. Syst. Softw. 136, 173–186.
Al-Sabbagh, K., Staron, M., Hebig, R., 2022. Predicting build outcomes in continuous integration using textual analysis of source code commits. In: Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering. PROMISE'22, pp. 42–51.
Alenezi, M., Banitaan, S., 2013. Bug reports prioritization: Which features and classifier to use? In: Proceedings of the 12th International Conference on Machine Learning and Applications. ICMLA'20, Vol. 2, pp. 112–116.
Almhana, R., Ferreira, T., Kessentini, M., Sharma, T., 2020. Understanding and characterizing changes in bugs priority: The practitioners' perceptive. In: Proceedings of the 20th International Working Conference on Source Code Analysis and Manipulation. SCAM'20, pp. 87–97.
Apache, 2023. Guidelines for creating a Jira ticket. URL https://infra.apache.org/jira-guidelines.html, Accessed: January 22, 2023.
Basili, V.R., 1992. Software modeling and measurement: The goal/question/metric paradigm. URL http://drum.lib.umd.edu/bitstream/1903/7538/1/Goal_Question_Metric.pdf.
Campbell, J.L., Quincy, C., Osserman, J., Pedersen, O.K., 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. Sociol. Methods Res. 42 (3), 294–320.
Chauhan, A., Kumar, R., 2020. Bug severity classification using semantic feature with convolution neural network. In: Computing in Engineering and Technology. Springer, pp. 327–335.
Cheng, C., Li, B., Li, Z.-Y., Zhao, Y.-Q., Liao, F.-L., 2017. Developer role evolution in open source software ecosystem: An explanatory study on GNOME. J. Comput. Sci. Tech. 32, 396–414.
Etemadi, V., Bushehrian, O., Akbari, R., Robles, G., 2021. A scheduling-driven approach to efficiently assign bug fixing tasks to developers. J. Syst. Softw. 178, 110967.
Feng, Z., Khomh, F., Ying, Z., Hassan, A.E., 2012. An empirical study on factors impacting bug fixing time. In: Proceedings of the 19th Working Conference on Reverse Engineering. WCRE'12, pp. 225–234.
Field, A., 2013. Discovering Statistics using IBM SPSS Statistics, fourth ed. Sage Publications Ltd., Singapore.
Gavidia-Calderon, C., Sarro, F., Harman, M., Barr, E.T., 2021. The Assessor's Dilemma: Improving bug repair via empirical game theory. IEEE Trans. Softw. Eng. 47 (10), 2143–2161.
Gökçeoğlu, M., Sözer, H., 2021. Automated defect prioritization based on defects resolved at various project periods. J. Syst. Softw. 179, 110993.
Habayeb, M., Murtaza, S.S., Miranskyy, A., Bener, A.B., 2018. On the use of hidden Markov model to predict the time to fix bugs. IEEE Trans. Softw. Eng. 44 (12), 1224–1244.
Hassan, A.E., 2009. Predicting faults using the complexity of code changes. In: Proceedings of the 31st International Conference on Software Engineering. ICSE'09, pp. 78–88.
Izadi, M., Akbari, K., Heydarnoori, A., 2022. Predicting the objective and priority of issue reports in software repositories. Empir. Softw. Eng. 27 (2), 50.
Kanwal, J., Maqbool, O., 2012. Bug prioritization to facilitate bug report triage. J. Comput. Sci. Tech. 27 (2), 397–412.
Kononenko, O., Baysal, O., Godfrey, M.W., 2016. Code review quality: How developers see it. In: Proceedings of the 38th International Conference on Software Engineering. ICSE'16, pp. 1028–1038.
Kumari, M., Singh, V.B., 2018. An improved classifier based on entropy and deep learning for bug priority prediction. In: Proceedings of the 18th International Conference on Intelligent Systems Design and Applications. ISDA'18, pp. 571–580.
Li, Z., Cai, G., Yu, Q., Liang, P., Mo, R., Liu, H., 2024. Replication package for "Bug Priority Change: An empirical study on Apache Projects". URL https://github.com/breezesway/BugPriorityChange.
Li, Z., Liang, P., Li, D., Mo, R., Li, B., 2020. Is bug severity in line with bug fixing change complexity? Int. J. Softw. Eng. Knowl. Eng. 30 (11&12), 1779–1800.
Menzies, T., Marcus, A., 2008. Automated severity assessment of software defect reports. In: Proceedings of the 24th IEEE International Conference on Software Maintenance. ICSM'08, pp. 346–355.
Motwani, M., Sankaranarayanan, S., Just, R., Brun, Y., 2018. Do automated program repair techniques repair hard and important bugs? Empir. Softw. Eng. 23 (5), 2901–2947.
Najafi, A., Rigby, P.C., Shang, W., 2019. Bisecting commits and modeling commit risk during testing. In: Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE'12, pp. 279–289.
Oliveira, E., Fernandes, E., Steinmacher, I., Cristo, M., Conte, T., Garcia, A., 2020. Code and commit metrics of developer productivity: a study on team leaders perceptions. Empir. Softw. Eng. 25, 2519–2549.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164.
Sharma, M., Bedi, P., Chaturvedi, K., Singh, V., 2012. Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: Proceedings of the 12th International Conference on Intelligent Systems Design and Applications. ISDA'12, pp. 539–545.
Stol, K.-J., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering. ICSE'16, pp. 120–131.
Tian, Y., Ali, N., Lo, D., Hassan, A.E., 2016. On the unreliability of bug severity data. Empir. Softw. Eng. 21 (6), 2298–2323.
Vieira, R.G., Mattos, C.L.C., Rocha, L.S., Gomes, J.P.P., Paixão, M., 2022. The role of bug report evolution in reliable fixing estimation. Empir. Softw. Eng. 27 (7), 1–39.
Yu, L., Tsai, W.-T., Zhao, W., Wu, F., 2010. Predicting defect priority based on neural networks. In: Proceedings of the 6th International Conference on Advanced Data Mining and Applications. AMDA'10, pp. 356–367.
Yuan, T., Lo, D., Xin, X., Sun, C., 2015. Automated prediction of bug report priority using multi-factor analysis. Empir. Softw. Eng. 20 (5), 1354–1383.
Yuan, W., Lu, S., Sun, H., Liu, X., 2020. How are distributed bugs diagnosed and fixed through system logs? Inf. Softw. Technol. 119, 106234.
Zou, W., Lo, D., Chen, Z., Xia, X., Feng, Y., Xu, B., 2018. How practitioners perceive automated bug report management techniques. IEEE Trans. Softw. Eng. 46 (8), 836–862.

Zengyang Li is currently an Associate Professor with the School of Computer Science, Central China Normal University, China. He received the Ph.D. degree in software engineering from the University of Groningen, The Netherlands, in 2015. He has published more than 60 papers in international journals (e.g., JSS, IST, TOSEM), conferences (e.g., ICSME, ICPC, ESEM, WICSA, MODELS, ASE), and Elsevier books. His main research interests are in software quality, software architecture, and mining software repositories.

Guangzong Cai is currently a Master Student with the School of Computer Science, Central China Normal University. He received the B.S. degree in computer science and technology from Yangtze University in 2021. His main research interests are in software bugs, bug priority, bug association, and mining software repositories.

Qinyi Yu is currently a software engineer in NIO, China. She received the Master. degree in software engineering from Central China Normal University in 2022. Her main research interests are in multi-language software systems, technical debt, and mining software repositories.

Peng Liang is a Professor of Software Engineering in the School of Computer Science, Wuhan University, China. He received a Ph.D. degree in software engineering from Wuhan University, China. His research interests lie in the areas of software architecture, design, and intelligent software engineering, with a strong emphasis on using empirical software engineering approaches. He received IEEE TCSE Distinguished Paper award at ICPC 2021. He is in the Editorial Board of Journal of Systems and Software, Elsevier, and a Young Associate Editor of Frontiers of Computer Science, Springer. He has published more than 100 articles in peer-reviewed international journals, conference and workshop proceedings, and books.

Ran Mo is currently an Associate Professor with the School of Computer Science, Central China Normal University, China. He received the Ph.D. degree in software engineering from Drexel University, USA, in 2018. He has published more than 30 papers in international journals (e.g., TSE, TOSEM, IST, JSS) and conferences (e.g., ICSE, ASE, ESEM, ICSME, ICPC, WICSA). His main research interests are in software architecture, Microservices, and software quality.

Hui Liu is currently an Associate Professor with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, China. She received the Ph.D. degree in discrete technology and production automation from the University of Groningen, The Netherlands, in 2013. She has published more than 50 papers in international journals, conferences, and IET books. Her main research interests are in complex networks, automatic control, software quality, and artificial intelligence.