IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received August 28, 2021, accepted September 21, 2021, date of publication September 28, 2021, date of current version October 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3116061

An Empirical Study on Using Multi-Labels for Issues in GitHub

JINDAE KIM[1], (Member, IEEE), AND SEONAH LEE[2], (Member, IEEE)
[1]Department of Computer Science and Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea
[2]Department of Aerospace and Software Engineering and Department of AI Convergence Engineering, Gyeongsang National University, Jinju 52828, Republic of Korea

Corresponding author: Seonah Lee (saleese@gnu.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) Grant by the Ministry of Education under Grant NRF-2018R1D1A1A02085551 and Grant NRF-2021R1A2C1094167, and in part by the Human Resources Development of Korea Institute of Energy Technology Evaluation and Planning (KETEP) Grant by the Ministry of Trade, Industry and Energy under Grant 20194030202430.

ABSTRACT On GitHub, one of the most successful services for software project hosting, labels have been used to represent various information and decisions about reported issues. However, previous studies on labels were limited to simple statistics or label recommendations for issue types. In this paper, we aim to provide a better understanding of labels and their usage in software development. We particularly focus on using multiple and custom labels on issues. To analyze label usage, we collected software project data and label usage information from GitHub. We then quantitatively investigated the performance of projects with multi-label features and qualitatively investigated the categories of multi-labels, and the usage of multi-labels based on these categories. Our analysis results show that multi-labels are common in the majority of software projects and that projects using multi-label features manage their issues more effectively. In addition, our analysis results reveal different types of information represented by labels, which are related to features, development, and issues. This study finds several facts that can be used for studies on issue management and thus that help develop labeling techniques to mitigate the burden of issue management.

INDEX TERMS Issue tracking system, issue management, issue labels, multi-labels, github, open source software, software maintenance.

I. INTRODUCTION

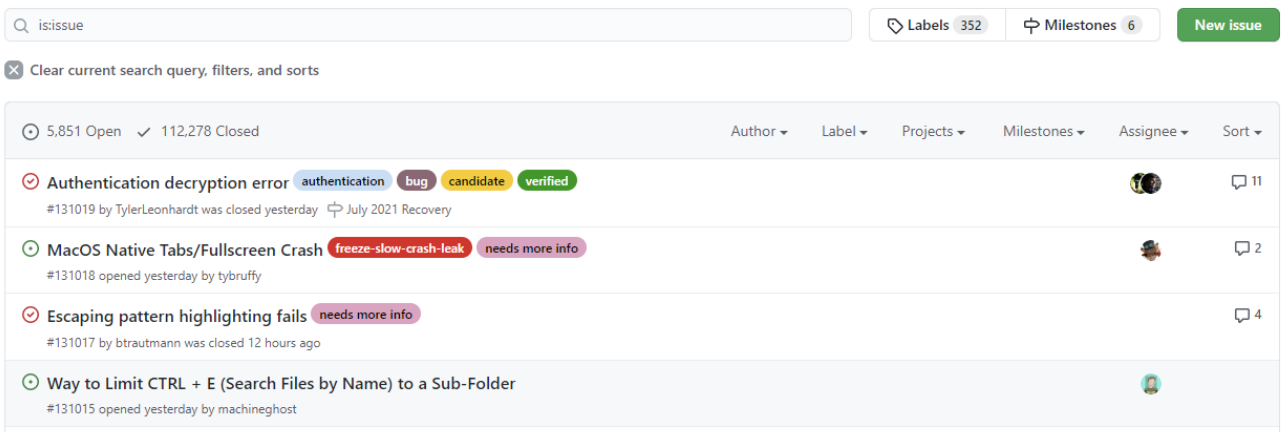
GitHub is one of the most successful, flourishing services for software project hosting with an issue management system. In the issue management system, GitHub provides support for *labels*, which can be used to represent various information and decisions about reported issues. Project contributors can freely attach any labels to express the same kind of information regarding their issues, as shown in Fig. 1 . Issue labeling can increase the visibility of issues and can facilitate contributors to resolve issues in less time.

The labeling system is the backbone of GitHub's issue management system. Due to such importance, there have been several studies to analyze and improve the labeling system. Some studies sought to analyze labels to reveal interesting facts on issue management [1], [2]. Other studies developed techniques to analyze and improve the labeling system, such as visualizing label usages [3] or automatically labeling reported issues [4].

However, there exist limitations in previous studies on the labeling system, and many questions are still unanswered. First, none of the previous studies considered aspect of using multiple labels for an issue. Different labels assigned to issues indicate different information or attention on the issues, which may affect issue management. Second, previous studies did not investigate custom labels. Custom labels can represent the information that developers would like to express for managing issues. On the other hand, custom labels might be one of the huge obstacles that interfere with label analysis on issue management. For instance, one of the popular labels 'bug', which represents the type of an issue, has many variants. Although GitHub provides the default label 'bug', many projects have their own labels, such as 'bugs'

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

Fig. 1 shows the labels attached to issues. Among them, *bug* is a default label that GitHub provides, while other labels are custom labels that project contributors have created. Issues have a different number of labels, because project contributors decide how many labels they will use for each issue. As shown in the upper right corner of the figure, the *vscode* project has 352 labels as of August 21, 2021.



5,851 Open ✓ 112,278 Closed

Authentication decryption error authentication bug candidate verified
#131019 by TylerLeonhardt was closed yesterday July 2021 Recovery 11

MacOS Native Tabs/Fullscreen Crash freeze-slow-crash-leak needs more info
#131018 opened yesterday by tybruffy 2

Escaping pattern highlighting fails needs more info
#131017 by btrautmann was closed 12 hours ago 4

Way to Limit CTRL + E (Search Files by Name) to a Sub-Folder
#131015 opened yesterday by machineghost

FIGURE 1. Example of labels attached to issues for the *vscode* project in GitHub.

or 'type:bug', to represent the same thing. To obtain more meaningful, interesting information, we need to understand the characteristics of custom labels.

In this study, we considered all these limitations and sought to answer questions that still remained unknown. To the best of our knowledge, this is the first attempt to analyze the labeling system of GitHub from the perspective of multi-labels and the first to analyze custom labels. We carefully designed the process to refine data from the GHTorrent dataset [5] and updated the data with recent information obtained from GitHub. The subjects we analyzed included 14,415 projects with 13 million issues and 0.3 million labels.

We first analyzed how many issues have multiple labels and how multiple labels affect issue management. Our analysis result shows that 90% of the projects have used multiple labels on their issues at least once. Moreover, projects with multi-label issues have a higher close rate (81%) than the other projects with no multi-label issues (68~74%). In terms of close time, the projects with multi-label issues have a longer average close time than the other projects without multi-label issues, since multi-label issues are often more difficult to resolve. However, if we compare the average close time of single-label issues only, then projects having multi-label issues have more than a month shorter close time (131.9 days) than projects having single-label issues without multi-label issues (173.6 days). These results indicate that projects using multi-labels have resolved their issues more effectively.

Next, we analyzed what kinds of custom labels are defined in projects, and how such labels have been assigned to issues. We qualitatively analyzed custom labels in three selected projects. Our analysis result shows that issues have up to 7~9 labels and frequently use 2~4 labels. Custom labels are usually defined in the *feature*, *development*, *issue* concepts. In particular, custom labels that are relevant to *issue status* are actively used. Meanwhile, custom labels that are relevant to *functionality* are actively used in two projects, but not in one project. When we investigated labels used together, we found that labels representing a bug were widely used in the three projects, but other labels frequently used together varied according to projects. These findings indicate that different kinds of labels work differently on issues during issue management.

We listed contributions of this study as follows.

- *We Design a Process to Collect and Refine Data From GitHub for Issue and Label Analysis:* We carefully refined projects from existing the GHTorrent dataset to improve the credibility of our study results and described the process in detail for similar future studies.
- *We Provide an Empirical Study Analyzing Multi-Labels' Effectiveness on Issue Management:* We compared issue management performance when no-label, single-label or multi-label was used. The result shows that projects using multi-labels have resolved their issues more effectively.
- *We Analyze How Labels Are Customized:* Unlike other previous studies on GitHub labels, we studied how labels have been customized. We devised label categories that can be used in future studies to aggregate custom labels into groups and to identify more information and characteristics at a high-level.

The remainder of sections will be organized as follows. First, Section III explains our empirical study design including research questions, data collection and analysis methods. Then Section IV presents analysis results and discusses their implications, and Section V provides further discussion about custom labels and label categorization. After that, Section II discusses related work. and Section VI discusses threats to validity of this study Finally, Section VII concludes the study with future work.

II. RELATED WORK

We classify our related work into three categories: empirical studies on GitHub, studies on issue labeling and classification, and empirical studies on issues.

A. EMPIRICAL STUDIES ON GitHub

There have been many empirical studies on GitHub, which have investigated various angles of software development and its analysis on GitHub.

Several studies on GitHub revealed the pitfalls of mining GitHub, and they affected our design of subject refinement and analysis. Kalliamvakou *et al.* investigated the advantages and disadvantages of mining GitHub [6], [7] after Bird *et al.*'s analysis on the advantages and disadvantages of Git [8]. They showed that the addressed biases could threaten the validity [7]. Likewise, Consentino *et al.* also discussed their concerns about the reliability of GitHub studies due to issues such as poor sampling techniques [9]. Munaiah *et al.* noted that GitHub repositories contain not only engineered software projects but also noisy contents that are irrelevant to software projects [10]. We reviewed these studies and considered and adapted their findings when we designed our subject refinement and experiments.

Two recent studies investigated forks and pull requests. Zhou *et al.* revealed that the concept of forks has changed from hard forks that are independent development branches to social forks that are temporary branches created for feature implementation [11]. Brown and Parnin investigated suggested changes associated with pull requests on the GitHub system [12]. They uncovered the facts that developers mainly suggested correcting errors, formatting and improving code, and modifying the non-functional part of the code. These studies targeted different subjects - forks and pull requests - unlike multi-labels on issues in our study. However, the findings in those studies helped us to understand software development community on GitHub and were useful for our analysis on issues and labels related to the development.

Other studies focus more on software development activities and communities on GitHub. Borges *et al.* explored the characteristics of popular GitHub repositories by assuming the star ratings of repositories as their popularities [13]. They analyzed the correlations of the star ratings and other characteristics, such as ages, commits, contributors, forks and more, and they found a strong correlation between stars and forks. Subramanian analyzed the first contributions of OSS developers by collecting the first pull requests of 3,501 developers [14]. He reported that about 30% of the changes were only on one to five lines of code, which were mainly modified or inserted. Although these studies explored different territories, they provide valuable information to understand software development activities on GitHub, similar to our study on multi-label usages.

B. STUDIES ON ISSUE LABELING AND CLASSIFICATION

There are studies that focus on issue labeling [1]–[3]. First, Izquierdo *et al.* noted that labeling is a basic classification mechanism for issue reports, but it is infeasible to infer relevant information from labels as issue reports increase [3]. To address this issue, they developed a GitHub Label Analyzer that visualizes label usages in a graph and label timeline in a tree form.

Liao *et al.* investigated the importance of behaviors related to issue management [1]. Similar to our study, they used the data collected from GitHub Archive and seven popular projects collected by using GitHub REST API. By analyzing the data, they found interesting facts such as that labeling issues have a positive impact on issue processing and the trends of label changes according to different development phases. While they found a positive impact of labels on issue reports, they did not focus on multi-labels, or how individual labels affected issues differently.

Cabot *et al.* studied the impact of issue labeling on open source projects [2]. With respect to label usage, they found that only 122,012 out of 3,737,038 projects (3.25%) attached labels to issue reports. They also found that the most commonly used label is 'enhancement'. The next most common labels are 'bug' and 'question'. These labels were sometimes used together. With respect to label influences, they found that labeled issues have a high percentage of being solved (more than 43.51%), compared to unlabeled issues (22.53%). While they identified commonly used labels, they did not investigate the multi-labels commonly used together.

In addition, researchers have steadily studied automatic classification of issue reports, since Herzig *et al.* manually categorized more than 7,000 issue reports and found that one-third of bug reports are irrelevant to bugs [15]. Researchers have proposed techniques to classify issue reports into two categories: bugs and non-bugs [16]–[23]. In addition, Kochhar *et al.* proposed classifying issue reports into thirteen pre-defined categories [24], and Fazayeli *et al.* proposed classifying issue reports into five categories [25]. Recently, Kallis *et al.* proposed an automatic labeling system, called TicketTagger [4]. TicketTagger predicts a label whenever an issue report is submitted to the GitHub issue tracking system and automatically attaches the predicted label to the submitted report. To predict a label relevant to an issue, TicketTagger preprocesses the issue report, expresses the issue reports as a vector, and classifies the issue report. Kallis *et al.* evaluated TicketTagger by applying it to balanced data with 10-fold cross-validation. The evaluation results show 78.1% to 82.2% precision and 76.3% to 87.4% recall. TicketTagger limits the labels to the three types of issues: Bug, Enhancement, and Question. These classification studies mainly focus on issue types, while our study explores the performance of projects with multi-labels and the kinds of custom labels on issues, not limiting issue types.

Our study differs from the previous studies [1], [2], [4], [16]–[21], [24], [25] since our study focuses on the usage of multi-labels. Our study goes one step further than previous studies [1], [2] by showing that the projects with multi-labels have a shorter resolution time than the projects without multi-labels. In addition, our study could be a basis for research on automatically assigning multiple labels to an issue report, which previous studies have not yet proposed.

C. EMPIRICAL STUDIES ON ISSUE DISCUSSION AND SUCCESS

There are two lines of work in empirical studies on issue reports. One line of work studied the discussions in issue reports [26]–[31], and the other co-related issue reports with project or issue success [32]–[35].

Among the studies on the discussions in issue reports, Krishna *et al.* worked on predicting the trends of bugs and enhancements based on the number of issue reports [26]. Hu *et al.* studied the multiple discussions of developers across several issue reports [27]. Hu *et al.* found that projects with multiple discussions tend to have a shorter resolution time than the others. Likewise, our study reveals that projects using multi-label features tend to have a shorter close time.

Four studies investigated or classified the comments of issue reports [28]–[31]. Arya *et al.* qualitatively classified 4,656 comments into 16 major types such as solution discussion and bug reproduction [29]. Huang *et al.* analyzed 154,493 issue reports that had questions (29.73%) and no questions (70.27%) [28]. Issue reports with questions have a longer elapsed time and a higher number of developers, comments and reassignments. Rath and Mäder investigated developers' communication patterns by analyzing 270,000 comments [30]. They found conversations for collaborations (44%), conversations for feedback (41%), and monologues (15%). Raman *et al.* developed a sentimental classifier to detect toxic conversations among developers [31]. By applying the classifier to 1,734,124 issues, they found that toxic conversations decreased from 2012 to 2018.

Our study differs from these studies since we address different subjects, issue comments versus issue labels. However, we noticed that label prediction approaches could utilize comments as one of the features to predict labels [4], [36]. Their studies on issue comments and our study on issue labels could be considered together to develop an effective mechanism for predicting labels on issue reports.

Among the studies that co-related issue reports with project or issue success, Bissyandé *et al.* defined project success as the popularity of a project and investigated the impact of the adoption of issue tracking systems on project success [32]. They found that one-third of projects maintain issue reports and that the projects with issues are maintained a half a year longer than other projects without issues. Kikas *et al.* suggested dynamic features such as the number of comments in an issue and contextual features such as the number of issues closed in two weeks [33]. They found that contextual features play an important role in predicting issue lifetime and that dynamic features contribute to the prediction less than contextual features but still complement static and contextual features. Ramírez-Mora *et al.* identified successful and unsuccessful issues [34]. They also found that there are key development phrases in the trend of issue reports. Zhou *et al.* studied the impact of bounties on addressing issues in GitHub [35]. They found that the timing of proposing bounties is important to address issue reports and that the higher the bounty is given, the more likely the issue reports are to be resolved.

Our study goes a different direction from these studies, since our study conducts an empirical study on multi-labeled issue reports. Furthermore, our study goes one step further from Bissyandé *et al.*'s [32] by showing that projects with labeled issues maintain a higher close rate and a shorter close time than projects with only unlabeled issues.

III. STUDY DESIGN

In this section, we will present research questions and explain study design, including data collection and analysis methods. We will use the terms repository and project interchangeably, as we consider one repository as one project in this study. The concept of a project could be broader than that of a repository. For example, the *dart-lang* project has several repositories such as *sdk* and *language*. However, it will not considerably affect our analysis, since a repository that belongs to a project can be considered as a sub-project, such as *dart-lang/sdk*.

A. RESEARCH QUESTIONS

We introduce research questions regarding multi-label usage and its influence on issue management performance and research questions investigating multi-labels in projects.

1) RQ1. MULTI-LABELS AND ISSUE MANAGEMENT PERFORMANCE

- *RQ1.1 How Popular Are Multi-Labels in Issue Management?* We investigated how many projects and issues used labels, before we analyzed issue management performance of projects and issues with respect to the usage of labels.
- *RQ1.2 Does Using Multi-Labels Affect Issue Management Performance of Projects?* Each project may have their own policy to assign labels to issues for management. We analyzed whether using multi-labels affected issue management performance of projects.
- *RQ1.3 Does Using Multi-Labels Increase Issue Management Performance of Individual Issues?* In a software project, there might be different issues with different numbers of labels. We investigated whether multi-label issues have differences in issue management performance compared to other issues with no labels or just one label.

2) RQ2. INVESTIGATION ON MULTI-LABELS IN PROJECTS

- *RQ2.1 How Many Labels do Issues Have in Projects?* We investigated how many labels are used for labeling issues, especially focusing on how many labels are attached to an issue.
- *RQ2.2 What Kinds of Custom Labels Are Defined in Projects?* Although GitHub provides default labels for issues, many projects define their own custom labels. We analyzed the kinds of custom labels that are defined in projects.
- *RQ2.3 What Kinds of Custom Labels Are Used in Projects?* Even if custom labels are defined in projects, those labels might not be used in projects. We analyzed the kinds of custom labels that are used in projects.

https://github.com/dart-lang

- *RQ2.4 Which Custom Labels Are Used Together?* We analyzed which labels are frequently assigned together to represent various information about issues.

B. DATA COLLECTION

To answer the research questions, we first collected issue reports of open source projects and their labels in GitHub. GitHub provides REpresentational State Transfer Application Programming Interface (REST API) v3 to access various information of repositories. GHTorrent is an enormous public archive that collected such information for years [5]. At the time we started this study, the latest GHTorrent dataset provided data collected until June 2019, which contained information of more than 125 million repositories.

However, we could not use the dataset for the following reasons. First, the latest GHTorrent dataset was not actually up-to-date, since we started to collect data for this study in March 2020. We found that some of the repositories in GHTorrent were deleted. Some of the repositories were migrated to other repositories; hence, we could not access recent data of those repositories based on GHTorrent's records. Second, we focused on issues and their labels in this study, and not all repositories had such information. As revealed in a previous study [7], many of the repositories in GitHub are personal or not even about software development, which makes them out of this study's scope. Last, there were inconsistencies among the information stored in GHTorrent. For instance, some of the issues' were recorded as closed earlier than their creation, which would have affected our study.

For these reasons, we went through refinement steps to select our subjects, and collected the latest data of subjects via GitHub REST API. From the GHTorrent dataset, we first listed all projects that were not forked from another project and had at least 210 issues. These projects were not simply copied from other repositories and thus managed by developers more actively. Using a minimum number of issues was necessary to secure sufficient material for our study, and 210 issues were from 99% percentile of the ranking in the unique number of issues of projects. For these filtered projects, we collected the latest information to remove unavailable or duplicate projects. Finally, we discarded projects with no language information returned by GitHub, which indicated that there was no detectable code; hence, it was more likely that they were not about software development.

Table 1 shows the simple statistics of our subjects. The final dataset we used includes 14,415 projects with more than 13 and 0.3 million issues and labels, respectively. Languages used in subjects included JavaScript (21%), Java (12%), Python (12%) and many others such as PHP, C++, C and Ruby. These subjects had a maximum of 12 years of development histories, and the average period of development is 5.43 years. Popular open source software projects such as Visual Studio Code, Elasticsearch, TypeScript, and NumPy are also included in the subjects. Hence, our subjects are software projects in various programming languages with sufficient development histories for the study.

The key information and data of this study are publicly available in our repository. This repository contains the information of the 14,415 projects as well as the issues and labels. It also provides intermediate data we produced and used during analysis, which were reported as figures and tables in Section IV. We did not include complete raw data obtained from GitHub API and GHTorrent, since they are already publicly available and not all of them were used in this study.

C. ANALYSIS METHODS

In this section, we explain how we analyzed the collected data to answer the research questions.

1) RQ1. MULTI-LABELS AND ISSUE MANAGEMENT PERFORMANCE

For RQ1, we considered the number of labels assigned to issues of projects. Based on the number of labels, we categorized projects and issues into three groups: *No-label*, *Single*, and *Multi*. For issues, we simply assigned issues without any labels to the no-label group, those with only one label to the single label group, and those with more than one label to the multi-label group. For projects, we used the existence of single-label and multi-label issues. If a project contained any multi-label issues, it was categorized in the multi-label group, and any remaining projects with at least one single-label issues were categorized in the single label group. The remaining projects were assigned to the no-label group. Throughout the paper, we used terms no-label, single-label and multi-label to indicate No-label, Single and Multi groups respectively. For instance, single-label issues are the issue with only one label, and project using multi-label means the project belonging to the Multi group.

We analyzed multi-label usage and issue management performance for the three groups. For RQ1.1, we counted the number of issues and projects for each group and analyzed them. For RQ1.2, we measured issue close rate and issue close time to evaluate issue management performance. Issue close rate, or close rate is the ratio of closed issues to all issues in each project group. For example, as shown in Fig. 1, the *vscode* project has 5,851 open issues and 112,278 closed issues. In the case, the close rate is the ratio of 112,278 closed issues to 118,129 total issues, which is 0.95. We computed the close rate for each project, and then we averaged the close rates of all projects in each group.

Issue close time, or close time is the duration between the creation time and close time of an issue. For example, the issue titled "Escaping pattern highlighting fails" in Fig. 1 was created at 2021-08-17 16:38:14 and closed at 2021-08-18 14:25:54. Then the duration is less than 24 hours, hence the

https://developer.github.com/v3/
http://ghtorrent.org

https://github.com/Jindae/issue-label-study

TABLE 1. Information of subjects.

Item	Number
# of collected projects	14,415
# of collected issues	13,241,194
# of collected labels	330,332
Avg. development period (years)	5.43
Max. development period (years)	12

close time is computed as 0 days. We first computed the average close time of each project as the project's close time, and then computed the average close time of all projects in each group. We also investigated the distribution of projects' close time. For RQ1.3, close rate and close time is straightforward, and we computed close rate and close time for each issue group.

2) RQ2. INVESTIGATION ON MULTI-LABELS IN PROJECTS

We investigated how labels were created and used in projects. GitHub provides nine default labels. In addition, developers can freely create labels to organize issues in GitHub, which we call *custom labels*. First, to select the target projects to manually investigate custom labels, we selected the top 100 projects having the most number of issues. Then we manually examined the projects and selected 10 candidate projects. To select the candidate projects, we checked the following conditions for each project.

- Is it a software project developed actively?
- Does it have more than ten contributors?
- Does it actively use the issue tracking system at our investigation time?
- Does it actively use multi-labels?

After selecting 10 projects, we re-examined the projects and selected three target projects that possessed relatively many labels. As a result, we selected the *vscode*, the *dart-lang/sdk* and the *typescript* projects.

We first investigated the number of labels which issues have for RQ2.1. We analyzed the labels attached to issues and created an issue distribution of the three projects according to the number of labels.

We then qualitatively analyzed the custom labels for the three target projects in detail for RQ2.2 and RQ2.3. To understand the concepts of labels, we adopted a qualitative analysis method, Grounded Theory [37]. We manually tagged labels in two steps. We first tagged each label by a word commonly used in labels if possible. If the label did not contain any commonly used word, we assigned a word that stands for the label. If we found it difficult to catch the meaning of the label, we referred to the description of the label and tried to understand its meaning. If the description was insufficient to understand the label, we checked the issues that had the label to understand its context. After assigning the first tag to each project, we added the second tag to each label based on the concepts formed from the first tagging. The second tag represents a more general concept than the first tag. The attached tags might be subjective. To reduce the subjectivity, two authors of this paper assigned the tags separately. Then, the authors reviewed the tags and resolved the conflicts that arose in the review with discussion. Labels were re-tagged if any problems were found during the discussion.

In addition, we investigated the multi-labels that were used together in the three target projects for RQ2.4. We first extracted the labels of each issue and converted them to a set of labels for the issue. Then, we applied an association rule mining technique by using WEKA [38]. We finally sorted the derived association rules by the number of co-occurrences and analyzed the set of labels most frequently used together.

Close times are integers computed by MySQL func., so 0 days is used in the case.

https://docs.github.com/en/free-pro-team@latest/github/managing-your-work-on-github/about-labels

https://github.com/microsoft/vscode

https://github.com/dart-lang/sdk

https://github.com/microsoft/TypeScript

IV. ANALYSIS RESULTS

In this section, we present our analysis results and discuss their implications.

TABLE 2. Projects and issues with no, single and multi labels.

Groups	Issues	Projects
No-label	6,012,700 (45.41%)	351 (2.43%)
Single	3,771,005 (28.48%)	989 (6.86%)
Multi	3,457,489 (26.11%)	13,075 (90.70%)
Total	13,241,194	14,415

A. RQ1. MULTI-LABELS AND ISSUE MANAGEMENT PERFORMANCE

1) RQ1.1 HOW POPULAR ARE MULTI-LABELS IN ISSUE MANAGEMENT?

Table 2 shows how many issues and projects belong to the three groups. The numbers in parentheses represent the ratio of each group with respect to the total number of issues and projects. Table 2 shows that 26.11% of the issues are multi-label issues, 28.48% are single-label issues, and 45.41% are unlabeled. No-label issues compose a higher portion of the issues since many of them might not yet be triaged. In terms of projects, the Multi group accounts for 90.70% of all projects. The Single and No-label groups occupy 6.86% and 2.43%, respectively. The majority of the projects have used multi-labels during issue management, although the ratio of multi-label issues is similar to that of single-label issues. It means that using multi-labels is popular in issue management, but not only about half of the labeled issues were assigned multi-labels.

Fig. 2 represents the issue distribution of Multi group projects. For each project, we computed the portion of the issues that belonged to the No-label, Single and Multi groups. In each box plot, a horizontal bar indicates the median, and a dot represents the average. As shown in Fig. 2, no-label issues compose slightly less than 50% (47.99%) of the issues on average. The average ratio decreases for single-label and multi-label issues, which are 31.18% and 20.82%, respectively.

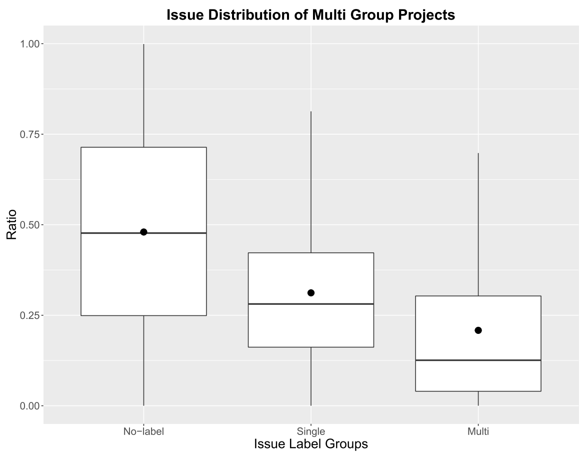


FIGURE 2. Issue distribution of multi group projects.

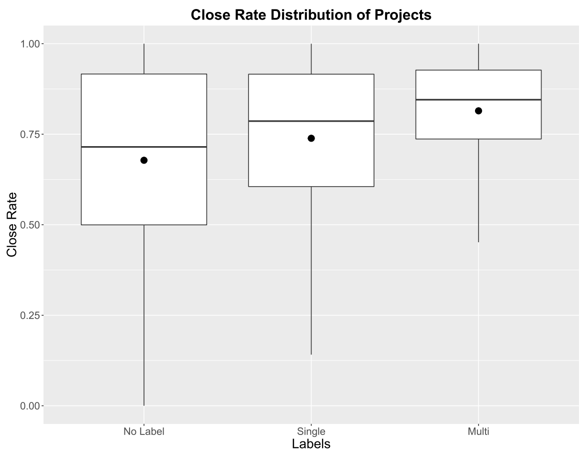


FIGURE 3. Close rate distribution of project groups.

TABLE 3. Close rate of project groups for different labels.

Issues	Close Rate		
	No-label group	Single group	Multi group
no-label	67.80%	74.12%	83.15%
single-label	N/A	70.68%	80.39%
multi-label	N/A	N/A	74.78%

These results show the reason why we categorized projects based on the existence, not based on the issue ratio. Even for Multi group projects, the ratio of no-label issues is high due to non-triaged issues. Additionally, not many of the issues are actually given more than one label. However, the low ratios of single-label or multi-label issues do not indicate that those projects have not utilized multi-labeling features. It is probable that many issues are not yet processed or are not worth tagging with multiple labels. Hence, we decided to consider the existence of single-label and multi-label issues for project categorization.

RQ.1.1 The percentage of projects that have multi-labels.
90% of the projects have used multi-labels at least once, but only 20% of the issues have multi-labels.

2) RQ1.2 DOES USING MULTI-LABELS AFFECT ISSUE MANAGEMENT PERFORMANCE OF PROJECTS?

Fig. 3 shows the distribution of close rate for each project group. We drew box plots in Section IV with the same configuration as follows. Every box plot represents its median with a straight line and its mean as a dot. Whiskers represent the minimum and maximum observed data within 1.5 times the Inter-Quartile Range (IQR) from above or below boxes. Since there are a great number of outliers, we omitted them for brevity.

In Fig. 3, the average close rates of No-label, Single, and Multi groups are 67.80%, 73.87% and 81.46%, respectively. Close rate tends to be higher in Multi group projects, demonstrated as a shorter box towards to the upper side. We conducted the KW (Kruskal Wallis) test, since close rate did not satisfy the normal distribution assumption. We then verified that the differences among the three groups were statistically significant (p-value < 0.05).

The results indicate that projects in Multi group have managed their issues more effectively. Compared to the other two groups, its average close rate is 13.66% (No-label) and 7.59% (Single) points higher, respectively. As shown in Fig. 3, the box plot of Multi group contracted upward compared to the other groups. This means that projects using multi-labels tend to have a higher close rate and thus that the projects handled reported issues effectively.

We further investigated that Multi group projects were indeed effective in their issue management. Table 3 shows the average close rate of no-label, single-label and multi-label issues in each group of projects. The average close rate of no-label and single-label issues in Multi group are 83.15% and 80.39% respectively. Single group projects have 74.12% no-label and 70.68% single-label close rates, and No-label group projects have only 67.80% close rate, which is lower than that of Multi group projects.

Based on these results (Table 3), Multi group projects have processed their no-label and single label issues more actively and effectively. On the other hand, projects which have never used single or multiple labels are less eager to manage their reported issues. Note that approximately half of the issues in Multi group projects are no-label issues (Fig. 2). Even if we consider that some of the no-label issues were not yet noticed, Multi group projects still show a higher close rate for no-label issues. Similar behavior was observed in a previous study [1], which showed that popular projects resolved almost all of their issues except for recently posted issues. Therefore, we may consider the usage of multi-labels as a sign of active issue management.

Fig. 4 presents box plots of the close time distribution. The close time of each project is the average of issues' close times in days. We also verified that the differences among the groups were statistically significant (p-value < 0.05) using the KW test.

In Fig. 4, Multi group tends to spend more time to close issues. The box plot of Multi group is higher than those of the other two groups. Both the average close time (dots) and

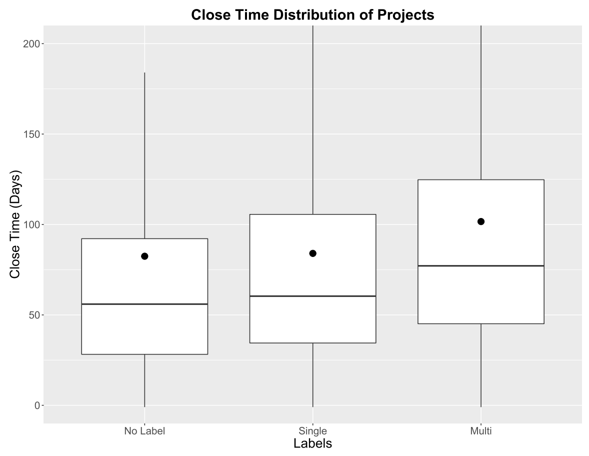


FIGURE 4. Close time distribution of project groups.

TABLE 4. Average close time of project groups for different labels.

Issues	Close Time (days)		
	No-label group	Single group	Multi group
no-label	82.7	79.0	67.4
single-label	N/A	173.6	131.9
multi-label	N/A	N/A	179.5

median close time (bars) were higher in the Multi group as well. The average close time of No-label, Single, and Multi group projects are 82.7, 84.2, and 101.6 days, respectively. On average, Multi group projects spent approximately 20 more days to close the issues.

However, these results do not indicate the inefficiency of issue management in Multi group projects. Table 4 shows the average close time of differently labeled issues for each project group. Only Multi group projects have multi-label issues which have a longer close time (179.5 days) than single or no-label issues, and this may lead to a longer close time of Multi group projects, unlike close rate. Additionally, the average close time of single-label issues in Multi group projects is 131.9 days, which is shorter than that of Single group projects (173.6 days). This is similar for no-label issues. Multi group projects have closed no-label issues in 67.4 days on average, which is more than 10 days shorter than 79.0 days of the Single group, and similar to 82.7 days of the No-label group. Therefore we can state that Multi group projects have managed their issues efficiently, although they have more time-consuming multi-label issues.

RQ.1.2 Issue management performance of projects that have multi-labels. *Projects with multi-labels have 7.59~13.66% higher close rate than other projects. However, projects with multi-labels spent more than 20 days to close the issues than other projects.*

3) RQ1.3 DOES USING MULTI-LABELS INCREASE ISSUE MANAGEMENT PERFORMANCE OF INDIVIDUAL ISSUES?

Table 5 represents the average close rate of the three issue groups. The No-label and Single issue groups have similar close rates, which are 84.80% and 84.48%, respectively. Multi issue group has the lowest close rate (79.62%), which is even lower than the total close rate. Although it is only about 5%p, the Multi group clearly has a lower close rate than the others.

TABLE 5. Close rate of issue groups.

Close Rate	No-label	Single	Multi	Total
Issues	84.80%	84.48%	79.62%	83.36%

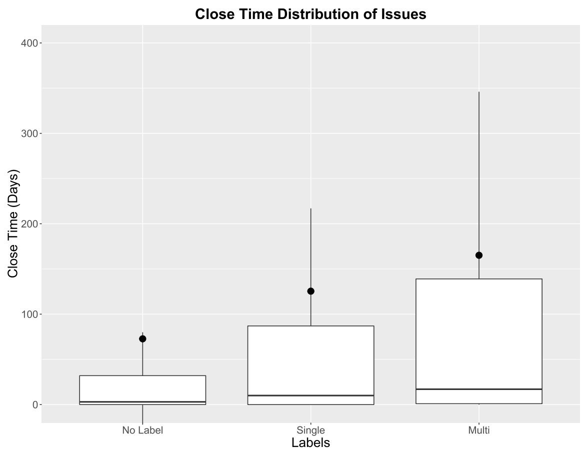


FIGURE 5. Close time distribution of issue groups in days.

These results imply that the issues marked with multiple labels tend to be more difficult to resolve. In RQ 1.2 (Table 3), we have already investigated the close rate of differently labeled issues in project-level. Note that the average close rate of multi-label issues at the project-level is 74.78%, which is lower than issue-level, since it is the average of project-level close rate. However, when we consider all issues together, the results are consistent with the project-level result, which indicates that issues with multiple labels are not closed easily compared to the no-label or single-label issues.

Fig. 5 shows the close time distribution of issue groups. Consistent with the close rate, multi-label issues have lower issue management performance. The average close time of Multi group was 165.2 days, which was approximately 93 days longer than that of the No-label group (72.7 days), and almost 40 days longer than the Single group (125.5 days).

This result indicates that multi-label issues require more time to resolve. The Multi issue group box plot in Fig. 5 spreads further than those of the other groups. We suspected that such a longer close time for multi-label issues was due to their difficulties and complications.

We further investigated the average number of comments, title length and body length of the issues. Issues with more comments or longer titles and bodies might indicate that these issues were more complicated and hence need more discussion. We found that multi-label issues tend to have more comments and longer titles and body lengths. Multi-label issues have more comments (4.05) than no-label (3.15) and single-label (3.56) issues. The average title length of the issues was also longer in multi-label issues (50.48) than those in no-label (44.63) and single-label (47.03) issues. The average body lengths were 1201.96 (Multi), 1139.44 (No-label) and 1018.05 (Single), respectively, which show that multi-label issues needed more characters to be described than the others. We also conducted the KW tests and confirmed that these differences were statistically significant (p-value < 0.05).

RQ.1.3 Issue management performance of issues that have multi-labels. *Multi-label issues have a lower close rate by 5% than the other issues, and the average close time of multi-label issues is approximately 40~93 days longer than that of other issues. Issues with multi-labels are more complicated and difficult to resolve.*

B. RQ2. INVESTIGATION ON MULTI LABELS

As explained in Section III-C2, we qualitatively analyzed labels of the three projects. We first analyzed the number of labels that issues have. We then showed what kinds of custom labels are defined, and what kinds of labels are mainly used for issue labeling. We finally reported the labels that were frequently used together for each project.

1) RQ2.1. HOW MANY LABELS DO ISSUES HAVE IN PROJECTS?

We analyzed the number of issues according to the number of labels over the three projects. Note that the vscode project had 274 labels and 92,286 issues, the dart-lang/sdk project had 208 labels and 41,864 issues, and the typescript project had 127 labels and 30,702 issues. Fig. 6 shows the issue distribution according to the number of labels attached to the issues in three projects.

First, Fig. 6a shows the number of issues according to the number of labels in the vscode project. In Fig. 6a, the number of issues with a single label is the highest over 20,000, while the number of issues with two or more labels is greater than the number of issues with a single label. The number of issues without labels is less than the number of issues with three or fewer labels but more than the number of issues with four or more labels.

Fig. 6b and Fig. 6c show similar trends to Fig. 6a. Nevertheless, the number of issues with a single label is the highest, but the number of issues with two or more labels is greater than or similar to the number of issues with a single label. Projects maintain up to 7~9 labels per issue, as shown in the figure.

RQ.2.1 The number of labels that issue have. *Issues have up to 7~9 labels. While one label is most commonly used for issues, two, three and four labels are also frequently used. In contrast, more than five labels are seldom used.*

2) RQ2.2. WHAT KINDS OF CUSTOM LABELS ARE DEFINED IN PROJECTS?

To understand what kinds of custom labels are defined, we analyzed the custom labels by tagging them in two steps. The results are shown in Table 6, Table 7 and Table 8.

In each table, the Classification column lists categories of labels. The Label Examples column shows label examples that are included in each category. Next, the #Labels column represents the number of labels in the category, and %Labels provides the ratio of labels in the category to the total labels. #Issues represents the number of issues associated with the category, and %Issues is the ratio of issues in the category to the total issues.

Table 6 presents the classification results for the labels of the vscode project. The labels of the vscode project are classified into 10 categories and 3 concepts. The first concept is related to features to which functionality, environment, language, and non-functionality categories belong. The second concept is related to development to which version, configuration and development phase categories belong. The third concept is related to issues to which issue status, issue type and issue priority categories belong.

When we analyzed the ratio of each category of labels in the vscode project, we found that the functionality category includes the most labels with 51.46% (see Table 6). The category that includes the second most labels is the issue status with 9.49%. The development phase includes a similar ratio of labels with 9.12%. The environment category follows with 8.39%.

Table 7 shows the results for the labels of the dart-lang/sdk project. The labels of the dart-lang/sdk project are classified into 11 categories and 3 concepts. The concepts are the same as in Table 6, but the categories are slightly different from Table 6. Different categories between Table 6 and Table 7 of the two projects are marked with '*', which are language for the vscode project in Table 6 and documentation and implementation details for the dart-lang/sdk project in Table 7. Two categories, language and implementation details, are classified into the feature concept, while documentation is classified into the development phase concept.

Regarding the ratio of each category of labels in the dart-lang/sdk project, the functionality category includes the most labels with 42.79% (see Table 7), which is the same as the analysis result of the vscode project. The category that includes the second most labels is implementation details with 18.27%, which is unique to the dart-lang/sdk project. The environment and issue status categories follow with 11.06% and 6.25%, respectively.

Table 8 shows the results for the labels of the typescript project. The labels are classified into 11 categories and 3 concepts. The concepts are the same as in Table 6 and Table 7. The categories are also similar to Table 6 and Table 7. The categories of language and documentation, which are different between Table 6 and Table 7 and reappear in Table 8, are still marked with '*'. The newly appearing categories in Table 8, author and festival, are marked with '**'.

Regarding the ratio of each category of labels in the typescript project, the functionality category includes the most labels with 29.92% (see Table 8), which is the same as the analysis result of the vscode and dart-lang/sdk projects. The category that includes the second most labels is issue

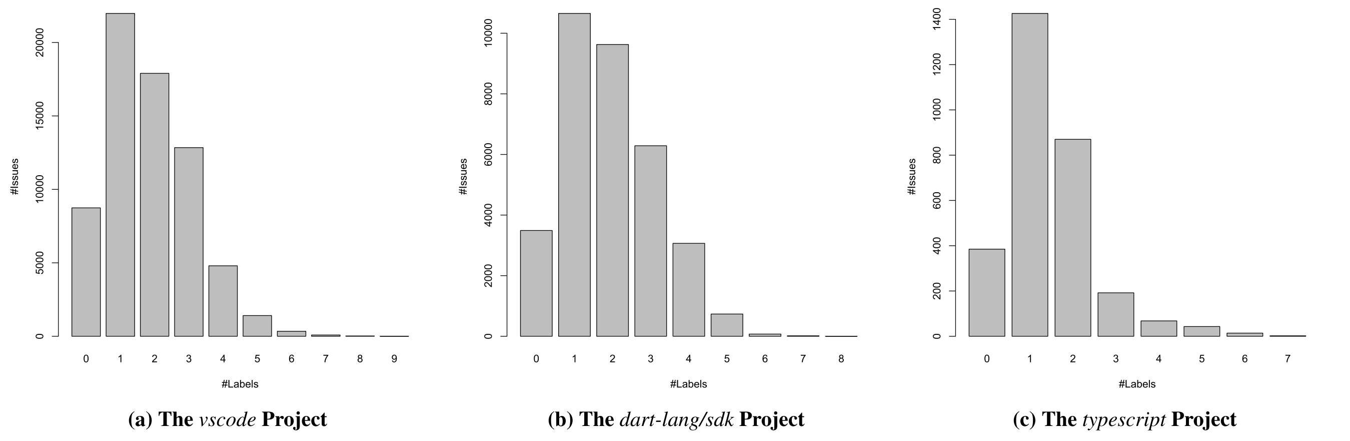


FIGURE 6. Issue distribution according to the number of labels.

TABLE 6. Qualitative classification of labels in *vscode* project.

Concept	Classification	Label Examples	#Labels	%Labels	#Issues	%Issues
feature	functionality	editor, view, file, font, intellisense, API, search, keybindings, workbench	141	51.46%	39,179	24.06%
	environment	device, OS, browser, extension	23	8.39%	10,414	6.40%
	language*	L10N, HTML, CSS, JSON, JavaScript, Node.js, PHP, WSL	12	4.38%	6,747	4.14%
	non-functionality	accessibility, authentication, performance, usability, internationalization	10	3.65%	3,580	2.20%
development	development phase	design, implementation, build, test, release, installation	25	9.12%	8,842	5.43%
	version	electron-2, electron-3, electron-4, electron-6, electron-8	15	5.47%	5,123	3.15%
	configuration	SCM, git, config, bot, update, website	13	4.74%	3,632	2.23%
issue	issue status	mitigated, confirmed, duplicate, insufficient info., invalid, wont fix, out-of-scope	26	9.49%	50,661	31.11%
	issue type	bug, duplicate, feature-request, question, comments	7	2.55%	32,500	19.96%
	issue priority	important, freeze-slow-crash	2	0.73%	2,143	1.32%
	Grand Total		274	100%	162,821	100%

status with 22.05%. The issue type and development phase categories follow with 12.60% and 10.24%, respectively.

RQ.2.2 The kinds of custom labels. *Custom labels are defined in three concepts that are relevant to features, development, and issues, respectively. In addition, the* functionality *and* issue status *are the categories where many custom labels are created across three projects.*

3) RQ2.3. WHAT KINDS OF CUSTOM LABELS ARE USED IN PROJECTS?

The ratio of custom labels and the ratio of using the custom labels can be different. Some labels are defined but not used, while one label can be used many times. To understand the usage of custom labels, we investigated the number of issues according to the label categories we identified in Section IV-B2. The results are shown in the #Issues and %Issues columns of Table 6, Table 7 and Table 8. Note that the total numbers of issues shown in Table 6 and Table 7 are larger than the numbers of issues we mentioned in Section III-C2 because one issue report could have more than one label.

The %Issues column of Table 6 shows that the labels in the *issue status* category are used the most at 31.11%. The labels in the *functionality* category are the second most used at 24.06%. The labels in the *issue type* category are the third most used at 19.96%. In contrast, the labels in the *development phase* and *environment* categories, which have a significant number of labels, are used at 5.43% and 6.40%, respectively.

The %Issues column of Table 7 shows that the labels in the *functionality* category are used a lot at 42.15% and the labels in the *issue type* are used at 16.29%. The column also shows that the labels in the *issue status* are used at 12.62% and the labels in the *environment* are used at 12.31%. Surprisingly, the labels in the *implementation detail* category, which maintains the second most labels, are used at only 6.27%.

The %Issues column of Table 8 shows a different distribution from the other projects. The labels in the *issue status*

TABLE 7. Qualitative classification of labels in *dart-lang/sdk* project.

Concept	Classification	Label Examples	#Labels	%Labels	#Issues	%Issues
feature	functionality	analyzer, area, customer, language, web, compiler	89	42.79%	37,017	47.15%
	implementation details*	library, package, part-of	38	18.27%	4,924	6.27%
	environment	browser, OS, VM, process	23	11.06%	9,661	12.31%
	non-functionality	security, soundness, time-out, message quality	4	1.92%	126	0.16%
development	version	dart1, dart2js	12	5.77%	267	0.34%
	configuration	merge, move, license	8	3.85%	1,249	1.59%
	development phase	epic, gardening, NNBD release, patch, design, testing	7	3.37%	1,504	1.92%
	documentation*	docs-API, docs-articles, docs-tools, docs-tutorials, documentation	5	2.40%	688	0.88%
issue	issue status	closed, cherry-pick, blocked, needs-info.	13	6.25%	9,906	12.62%
	issue type	bug, code-health, enhancement, performance, question, task	8	3.85%	12,792	16.29%
	issue priority	crash	1	0.48%	370	0.47%
	Grand Total		208	100%	78,504	100%

TABLE 8. Qualitative classification of labels in *typescript* project.

Concept	Classification	Label Examples	#Labels	%Labels	#Issues	%Issues
feature	functionality	domain, editor, API	38	29.92%	2,922	5.80%
	language*	JavaScript, typescript	7	5.51%	1,532	3.04%
	environment	tool, external, visual studio	4	3.15%	972	1.93%
	non-functionality	performance, internationalization	2	1.57%	104	0.21%
development	development phase	design, experiment, lib update, planning, new release	13	10.24%	2,136	4.24%
	version	ES6, ES7, ES2017, ES2018, ES2019	6	4.72%	160	0.32%
	documentation*	docs, docs meeting notes	3	2.36%	330	0.66%
issue	issue status	fixed, help wanted, needs investigation, needs more info., needs proposal	28	22.05%	20,645	40.99%
	issue type	suggestion, bug, enhancement, first issue	16	12.60%	19,414	38.55%
	issue priority	crash, moderate, high, critical	7	5.51%	1179	2.34%
etc.	author**	author:contributor, author:team	2	1.57%	919	1.82%
	festival**	pursuit fellowship	1	0.79%	51	0.10%
	Grand Total		127	100%	50,364	100%

category are the most used at 40.99% and the labels in the *issue type* category are the second most frequently used at 38.55%. Interestingly, the labels in *functionality* are used at only 5.80%.

***RQ.2.3 The kinds of custom labels used in projects.** All of three projects used the labels in the* issue status *category. Regarding the labels in the* functionality *category that two projects used the most, another project did not use the labels in the category much.*

4) RQ2.4. WHICH CUSTOM LABELS ARE USED TOGETHER IN PROJECTS?

We also investigated which custom labels are used together by mining association rules in the labels that are attached to issues together. As a result, Table 9 shows the top 5 most frequent multi-labels in the target projects.

As shown in the left column in Table 9, the labels 'verified' and 'bug' were most frequently used together 9,405 times in the vscode project. The labels 'verified' and 'feature-request' were the next most frequently used together. The labels 'verified' and 'verification-needed', 'bug' and 'debug', and 'feature-request' and '*out-of-scope' were also frequently used together. In the project, the top-5 multi-labels include 'bug', 'verified', 'feature-request', and other labels.

As shown in the middle column in Table 9, in the dart-lang/sdk project, the labels 'type-bug' and 'area-analyzer' were the most commonly used 2,154 times, followed by 'area-analyzer' and 'P2' 1,630 times, and 'type-bug' and 'P2' 1,368 times. In the project, the top 5 multi-labels include 'type-bug', 'area-analyzer', 'area-library', 'P2', etc.

As shown in the right column in Table 9, the labels 'awaiting more feedback' and 'suggestion' were the most frequently used together 203 times in the typescript project. The next frequently used labels are 'in discussion' and 'suggestion'. In the project, the top-5 multi-labels include 'bug', 'suggestion' and various other labels.

***RQ.2.4 Custom labels used together in projects.** The 'bug' relevant labels are included by all the projects. However, other labels are specialized for projects: 'verified' for the vscode project, 'area-analyzer' for the dart-lang/sdk project and 'suggestion' for the typescript project.*

V. DISCUSSION

We additionally investigate common label usage across all of our subjects, and discuss how the findings of our empirical studies help automate multi-labeling for issues.

TABLE 9. Top 5 most frequent multi-labels in the three projects.

	Labels	Freq.		Labels	Freq.		Labels	Freq.
vscode	bug, verified	9,405	dart-lang/sdk	area-analyzer, type-bug	2,154	typescript	awaiting more feedback, suggestion	203
	feature-request, verified	1,539		area-analyzer, P2	1,630		in discussion, suggestion	98
	verification-needed, verified	1,482		P2, type-bug	1,368		source:telemetry, bug	42
	bug, debug	1,332		area-library, library-io	1,021		help wanted, domain:lib.d.ts, bug	28
	feature-request, *out-of-scope	1,289		area-library, type-enhancement	976		effort:moderate, fix available, bug	18

TABLE 10. Top 5 most frequent labels used in many projects.

Rank	Label State	Frequency	Projects
1	bug	643,682	10,608
2	enhancement	404,458	8,603
3	question	231,245	7,089
4	**feature**	79,207	1,968
5	duplicate	66,537	4,942

A. COMMON LABEL USAGE

We inspected which label states were used frequently in all our subjects. A label state is a set of labels assigned to an issue, which consider single or multiple labels altogether. Table 10 shows the top 5 most frequent label states in descending order of their occurrences. The Frequency column indicates the number of occurrences, and The Projects column represents the number of projects in which each label state was observed. In the results, all frequent label states have one single label.

Overall, the majority of the frequent label states consist of one single label from nine default labels offered by GitHub. We found that the 'feature' label, which is the only non-default label indicating feature requests, is frequently used in almost two thousand projects. We first thought that 'feature' was another form of the default label 'enhancement'. However, it seems that developers want to distinguish them since they were willing to create a new label to use it, although there is a similar default label. Although we found an exception, our investigation on frequent label states mostly confirms that default labels are popular.

One of the reasons that default labels are captured in the top 5 most frequent label states is that there are many variations in custom labels that actually represent the same information. For instance, labels indicating low priority appear with different special characters (e.g., 'priority:low', and 'priority-low') and abbreviations (e.g., 'p: low', 'p3: low', and '[pri] low'). This is similar to the case in which two code snippets have the same functionality but differ in text due to coding style. Hence, we need more sophisticated methods to aggregate such custom labels.

B. DISCUSSION ON OUR EMPIRICAL FINDINGS

The results of our empirical study reveal several facts that can be used for studies on issue management.

First, our results on RQ1 show that the majority of projects - about 90% - have used multi-labels, and they have shown better issue management performance. Although the overall close time is longer, Multi group projects have a higher close rate, and shorter close time for no-label and single-label issues. Of course, there is no hard evidence that using multi-labels actually improves issue management performance. However, we can consider the usage of multi-labels as an indicator of projects with effective and efficient issue management, which are worth studying.

At this point, we could think of what kinds of multi-labels could be desirable for issue management. Through statistics, RQ2 reports that the number of issues with one label is the largest, while the number of issues with 2~4 labels is higher than the number of issues without labels. This indicates that 2~4 labels were carefully added to issues. Based on these facts, we suggest considering 2~4 labels in the multi-label automation study. We also suggest considering a single label.

Next, RQ2 investigated what kinds of custom labels are defined and used. We found that many labels for features, development, and issues are defined and that all three target projects utilized the labels that are relevant to issue status and type. This information can be helpful to future studies on issues and labels, since we can aggregate issues and labels and analyze them based on such categories.

We further analyzed issue management performance for issues with different kinds of labels in issue concept, since these labels appeared on all three projects and were used similarly. We first checked the close rate of issues with status labels and the others. The issues with status labels have a higher close rate (79.61%) than the other issues (72.67%). This indicates that using issue status labels can be helpful to handle issues, but automatic labeling techniques often focus on issue types only. Moreover, issue priority labels were seldom used in all three projects, but the close time of issues with priority labels is much shorter (171.9 days) than that of the other issues (202.4 days). It is likely that priority labels are mostly used when issues are urgent. However, this increases the importance of assigning priority labels, and it is worth developing a method to help such labeling and reduce the burden of developers for improved issue management.

VI. THREATS TO VALIDITY

Internal validity threats of this study mainly come from our subject refinement and analysis methods.

First, we did not use all the projects we collected, but instead refined the subjects that were more appropriate for our study. Due to the refinement, we might have introduced a bias in our dataset; hence, the results may be affected. However, without careful refinement, the results drawn from the unrefined dataset would have been more distorted since they contained too much inconsistent, outdated, and irrelevant information. We carefully designed our refinement steps to filter out the projects that are not software projects. All the steps were necessary to acquire projects which had enough data (i.e., issues) to be analyzed.

Second, we divided projects into three groups based on the existence of single and multiple labels. There are different ways to group projects such as using the ratio of single-label and multi-label issues in each project. If we used different grouping methods, the results and their implications might be different from the results of this study. However, this does not mean that our results are incorrect or meaningless. They still reveal interesting characteristics of projects using multi-labels. Additionally, the issue distribution results in Section IV-A1 show that it is inevitable to categorize projects based on existence, not based on issue ratio, due to the natural distribution of differently labeled issues.

Third, we conducted a qualitative analysis to understand custom labels, so the findings about the custom labels could be subjective. Our classification of labels in the three selected projects and label categorization results could be biased by our knowledge and experience. Moreover, we had to speculate about labels based on limited information such as labels themselves, the issues they were assigned, or their descriptions which were often unavailable. These are all threats that could undermine the validity of our analysis results. However, we tried our best to be objective in our judgment about custom labels.

Last, although we showed differences in issue management performance related to label usage, there exist other factors that affect the performance. For instance, a software project managed by more experienced senior software engineers may have better issue management. Similarly, functionalities of software, size of code, complexity of code, development methods and processes, tools, individuals in development or management teams, ownership of software, and many others can affect the issue management. Since this study is not based on a controlled experiment, we cannot control all these factors or variables. However, the goal of this study is not identifying all the factors which affect the issue management performance, but analyzing whether label usage is related to the performance. To that extent, we conducted statistical tests and confirmed that there exist statistically significant differences rather than random coincidental differences.

External threats of validity to this study are mostly from our limited access to data. Although we studied over 14K projects, 13 million issues and 0.3 million labels, we only collected publicly available data from a limited period of time. Since GitHub is also popularly used by private entities or enterprises, our findings about labels might not be valid for such cases. Also, even if we carefully refined projects, there is a possibility that our subjects are not representative of all software projects hosted by GitHub. However, our findings are still valid for software development projects that have managed issues to a certain degree.

VII. CONCLUSION

In this study, we investigated how developers have used multi-labels, a GitHub's distinctive feature of issue management. We collected and refined public data regarding issues and labels from GitHub for the study. With the collected data, we analyzed how multi-labels have influenced issue management performance, and how labels have been customized and used. Our analysis results include interesting findings regarding issues and labels. First, many software projects have exploited the multi-label feature of GitHub, and projects using multi-labels managed their issues more effectively. Second, although default labels are the most frequent ones, there are still various custom labels used by many projects. We grouped the custom labels and found that many of the labels are defined in the *feature, development,* and *issue* concepts and used together to represent the type, status and other characteristics of issues.

Based on the results, we understood that GitHub's multi-label features have been used quite actively, but there are some opportunities for improvement. First, a larger-scale study on the kinds of multi-labels could be conducted. However, since there are many variants in custom labels, we should develop a method to normalize the variants before the study. Additionally, since developers freely define custom labels, it is difficult to find the meanings of various custom labels automatically. We thus should find an automatic method to capture the meanings of labels and to classify them. Next, we need to analyze the multi-labels used together in more projects. To conduct such a study, we should adopt or develop an appropriate method to identify the kinds of multi-labels. Based on the method, we could more thoroughly obtain evidence for the facts revealed in this study. Through the process, it will be possible to suggest a more effective method for managing issues and an automatic labeling technique for issue management.

REFERENCES

[1] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in GitHub using visualization techniques," *IEEE Access*, vol. 6, pp. 24003–24015, 2018.

[2] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software projects," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 550–554.

[3] J. L. C. Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, and J. Cabot, "GiLA: GitHub label analyzer," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 479–483.

[4] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2019, pp. 406–409.

[5] G. Gousios, "The GHTorent dataset and tool suite," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, Piscataway, NJ, USA, May 2013, pp. 233–236.

[6] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*, New York, NY, USA, 2014, pp. 92–101.

[7] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining GitHub," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2035–2071, Oct. 2016.

[8] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *Proc. 6th IEEE Int. Work. Conf. Mining Softw. Repositories*, May 2009, pp. 1–10.

[9] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A systematic mapping study of software development with GitHub," *IEEE Access*, vol. 5, pp. 7173–7192, 2017.

[10] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Softw. Eng.*, vol. 22, no. 6, pp. 3219–3253, Dec. 2017.

[11] S. Zhou, B. Vasilescu, and C. Kästner, "How has forking changed in the last 20 years? A study of hard forks on GitHub," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 445–456.

[12] C. Brown and C. Parnin, "Understanding the impact of GitHub suggested changes on recommendations between developers," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Sacramento, CA, USA, Nov. 2020.

[13] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of GitHub repositories," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2016, pp. 334–344.

[14] V. N. Subramanian, "An empirical study of the first contributions of developers to open source projects on GitHub," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng., Companion*, Jun. 2020, pp. 116–118.

[15] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 392–401.

[16] S. J. Sohrawardi, I. Azam, and S. Hosain, "A comparative study of text classification algorithms on user submitted bug reports," in *Proc. 9th Int. Conf. Digit. Inf. Manage. (ICDIM)*, Sep. 2014, pp. 242–247.

[17] N. Pandey, D. Kumar, S. Abir, and H. Amitava, "Automated classification of software issue reports using machine learning techniques: An empirical study," *Innov. Syst. Softw. Eng.*, vol. 13, no. 4, pp. 279–297, 2017.

[18] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Nov. 2017, pp. 121–130.

[19] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? Bug report classification using N-gram IDF," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 534–538.

[20] N. Pandey, A. Hudait, D. K. Sanyal, and A. Sen, "Automated classification of issue reports from a software issue tracker," in *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*. Singapore: Springer, 2018, pp. 423–430.

[21] Y. Zhu, M. Pan, Y. Pei, and T. Zhang, "A bug or a suggestion? An automatic way to label issues," 2019, *arXiv:1909.00934*. [Online]. Available: https://arxiv.org/abs/1909.00934

[22] F. Elzanaty, C. Rezk, S. Lijbrink, W. van Bergen, M. Cote, and S. McIntosh, "Automatic recovery of missing issue type labels," *IEEE Softw.*, vol. 38, no. 3, pp. 35–42, May 2021.

[23] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empirical Softw. Eng.*, vol. 25, no. 6, pp. 5333–5369, Nov. 2020.

[24] P. S. Kochhar, F. Thung, and D. Lo, "Automatic fine-grained issue report reclassification," in *Proc. 19th Int. Conf. Eng. Complex Comput. Syst.*, Aug. 2014, pp. 126–135.

[25] H. Fazayeli, S. M. Syed-Mohamad, and N. S. Md Akhir, "Towards autolabelling issue reports for pull-based software development using text mining approach," *Proc. Comput. Sci.*, vol. 161, pp. 585–592, Jan. 2019.

[26] R. Krishna, A. Agrawal, A. Rahman, A. Sobran, and T. Menzies, "What is the connection between issues, bugs, and enhancements?" in *Proc. 40th Int. Conf. Softw. Eng., Softw. Eng. Pract.*, May 2018, pp. 306–315.

[27] D. Hu, T. Wang, J. Chang, G. Yin, and Y. Zhang, "Multi-discussing across issues in GitHub: A preliminary study," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 406–415.

[28] Y. Huang, D. A. da Costa, F. Zhang, and Y. Zou, "An empirical study on the issue reports with questions raised during the issue resolving process," *Empirical Softw. Eng.*, vol. 24, no. 2, pp. 718–750, Apr. 2019.

[29] D. Arya, W. Wang, J. L. C. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 454–464.

[30] M. Rath and P. Mäder, "Request for comments: Conversation patterns in issue tracking systems of open-source projects," in *Proc. 35th Annu. ACM Symp. Appl. Comput.*, Mar. 2020, pp. 1414–1417.

[31] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng., New Ideas Emerg. Results*, Jun. 2020, pp. 57–60.

[32] T. F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. L. Traon, "Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2013, pp. 188–197.

[33] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in GitHub projects," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, May 2016, pp. 291–302.

[34] S. L. Ramírez-Mora, H. Oktaba, and H. Gómez-Adorno, "Descriptions of issues and comments for predicting issue success in software projects," *J. Syst. Softw.*, vol. 168, Oct. 2020, Art. no. 110663.

[35] J. Zhou, S. Wang, C.-P. Bezemer, Y. Zou, and A. E. Hassan, "Studying the association between bountysource bounties and the issue-addressing likelihood of GitHub issue reports," *IEEE Trans. Softw. Eng.*, early access, Feb. 17, 2020, doi: 10.1109/TSE.2020.2974469.

[36] J. M. Alonso-Abad, C. López-Nozal, J. M. Maudes-Raedo, and R. Marticorena-Sánchez, "Label prediction on issue tracking systems using text mining," *Prog. Artif. Intell.*, vol. 8, no. 3, pp. 325–342, Sep. 2019.

[37] K. Charmaz, *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. Newbury Park, CA, USA: Sage, 2006.

[38] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

JINDAE KIM (Member, IEEE) received the B.S. degree in physics and computer science and the M.S. degree in computer science and engineering from Seoul National University, South Korea, in 2009 and 2011, respectively, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, in 2019. He is currently an Assistant Professor with Computer Science and Engineering Department, Seoul National University of Science and Technology. His research interests include automatic program repair, mining software repositories, and software evolution.

SEONAH LEE (Member, IEEE) received the B.S. and M.S. degrees in computer science and engineering from Ewha Womans University, in 1997 and 1999, respectively, the M.S.E. degree from the School of Computer Science, Carnegie Mellon University, in 2005, and the Ph.D. degree from the School of Computer Science, KAIST, in 2013. She worked as a Software Engineer with Samsung Electronics, from 1999 to 2006. She worked as a Research Professor at KAIST, from 2014 to 2015. She is currently an Associate Professor with the Department of Aerospace and Software Engineering and the Department of AI Convergence Engineering, Gyeongsang National University. Her research interests include software evolution, documentation updates, requirement traceability, software architecture, and data mining.