Issue Labeling Dynamics in Open-Source Projects: A Comprehensive Analysis

Joselito Jr
Federal University of Bahia
Salvador, Bahia, Brazil
joselito.mota@ufba.br

Lidia P. G. Nascimento
Federal University of Bahia
Salvador, Bahia, Brazil
lidianascimento@ufba.br

Alcemir Santos
State University of Piauí
Piauí, Brazil
alcemir@prp.uespi.br

Ivan Machado
Federal University of Bahia
Salvador, Bahia, Brazil
ivan.machado@ufba.br

Abstract

Open-source repositories play a vital role in modern software development, facilitating collaboration and code sharing among developers worldwide. In this study, we investigate the usage of labels in GitHub repositories to understand their impact on the issue resolution process and project management. We employ data mining techniques to gather a dataset comprising 10,673,459 issues from 13,280 repositories hosted on GitHub's featured topics list. Our study design involves four phases: repository selection, mining repository issues, pre-processing issues' components, and data processing to address research questions (RQs). The first RQ focuses on the frequency and usage of standard and custom labels in repositories. The second and third RQs delve into the average time for labeling issues and defining the triage phase from labeling practices. We found that 73.14% of repositories employ issue labeling, with most labeling activity concentrated before the 100th day since issue opening. This rapid labeling process is often followed by a structured label change pattern, potentially corresponding to specific issue phases like triage, implementation, or change validation. Analyzing time intervals between label changes, we observed that most issues undergo triage within 1 to 100 days, with labels prioritized based on their frequency in the resolution process. Our analysis sheds light on labels' significance in organizing and classifying issues through a systematic triage process within open-source repositories. Labels serve as social and technical elements, contributing to enhanced organization, identification, implementation, and validation of code changes. These findings provide valuable insights into the effective management and maintenance of open-source projects, aiding developers and project managers in optimizing issue resolution processes. The results and scripts from our study are available in the supplementary material repository for further exploration and reference by the software engineering community.

Keywords

Open-source Repositories, Issue, Issue labeling, Defect, Triage, Issue Life Cycle

1 Introduction

The open-source movement has experienced considerable growth and evolution, impacting both community dynamics and software development practices [31]. While the advantages of open-source projects are widely acknowledged, including flexibility, agility, and speed, managing these projects presents notable challenges. Project repositories must have robust infrastructure to host source code effectively and allow users, particularly software developers and testers, to report detailed information regarding identified issues [3].

Establishing clear communication channels to convey existing project information and planned changes is crucial for seamless evolution. Issue tracking systems (ITS) play an important role in this aspect, providing a platform to report issues with comprehensive details [13, 33, 34]. These systems streamline the issue lifecycle process and facilitate user collaboration, allowing users to report bugs, change requests, potential software improvements, and more. Users can also assign responsibility for addressing specific issue reports and exchange messages within the context of an issue. Employing ITS systematically is a key element for software maintenance and evolution in modern software engineering [1].

A well-structured issue report is crucial in helping developers understand the root cause of a reported problem and find an appropriate solution efficiently. In contrast, an incomplete issue report can pose expressive challenges for developers in a software project. Insufficient information or incorrect classifications can lead to confusion among software developers regarding the likely causes of issues [5, 7, 8].

An effective issue report should include essential elements such as a clear title, detailed body describing the problem, and relevant comments providing additional context or updates. Furthermore, many repositories allow users to add *labels* to categorize and provide contextual information about the issue [12]. These labels can enhance communication and streamline the issue triage process, aiding in understanding the problem's nature and identifying the appropriate developer to address it.

The practice of labeling involves assigning customized words or brief definitions to aid in quickly and objectively perceiving circumstances within social or management processes [36, 37]. It has become a common practice in open-source repositories [9], and there are distinct characteristics when labels are created either by software developers or automated algorithms [32]. Discussions regarding issue labeling have gained increasing importance in recent times [15, 19, 20, 24, 35, 39]. Contributions and repositories of

varying sizes and characteristics utilize these mechanisms to manage and involve the community in the progress of the repository [23, 40].

Despite its recognized importance, there remains a significant knowledge gap regarding the impact of issue labeling in open-source repositories. To address this gap in the literature, our work studies the utilization of labels by software developers, explores the associations between labels, examines the duration of each issue resolution phase, and investigates the relationships between each phase of the defect cycle, seeking patterns in label usage across these phases. Through these investigations, we aim to uncover label patterns associated with each phase of the issue's life cycle and analyze the time developers allocate to each phase. Our research tackles the following research questions (RQs):

RQ1: How often do developers use labels in open-source repositories? This RQ aims to understand the frequency of this practice and the number of repositories that either adopt standard labels or customize them to create their own. We leverage the count of customized labels in the repositories to identify the most commonly used labels and to measure their popularity.

RQ2: What is the average time developers take to label issues during the issue resolution? This RQ investigates the average time between the addition of new labels after previous labeling events. By examining this time gap, we can infer the duration it takes for a contributor to provide additional information relevant to issue resolution through labeling. This pattern of adding information can help characterize phases of issue resolution, such as the triage phase. Figure 1 illustrates the issue triage time considered in this RQ, starting from the period of Label 1 to the addition of Label 2.

RQ3: Can we define the triage phase based on labeling practices? This RQ investigates the triage time based on the presence of triage-related labels. We analyze the time it takes for an issue to go through triage from its opening until the last triage label is applied in the event sequence. Therefore, the triage time for an issue is characterized by the time it was opened and the time of the last triage-related label added. Figure 2 illustrates the issue triage time considered in this RQ, with a blue circle representing the opening time and Triage Label 3 being the last applied. Once the triage labels are categorized, the triage time can be defined based on the last label observed in the issue resolution process.

To achieve the stated objective and address the defined research questions, we conducted a characterization study focusing on labels in GitHub repositories. We gathered data from 10,673,459 issues across 180 GitHub featured topics [16] from a total of 13,280 repositories. The research package, including the issues database, results tables, and scripts utilized in this study, is accessible in the supplementary material repository.[1]

2 Background

A software repository typically comprises several visible components, including source code, a well-structured issue tracker with a defined life cycle, developers' messages, user messages, and other related elements [33]. Additionally, it involves community communication mechanisms that gather feedback from the community regarding changes and improvements made by the team of contributors. Practices involving community engagement, continuous integration, version history, and issue management represent essential dimensions of a comprehensive software repository [30].

Issue trackers are software solutions utilized by software repositories to systematically record all software behavior through reporting, encompassing various aspects of the repository such as code, verification, validation, documentation, and more. It is crucial to emphasize the construction of well-structured and effective communication management, as it plays a fundamental role in preventing new software problems, including the introduction of new defects [6, 29, 42]. User feedback, community collaboration, and maintainability are vital aspects for repositories, and users of issue trackers on platforms like GitHub should also adhere to these principles [13, 20, 34, 40].

2.1 The life cycle of an issue and its triage process

Various representations exist for the life cycle of an issue, as argued by Anvik et al. [4]. Issue tracking systems commonly depict possible states of an issue such as *new*, *resolved*, and *closed* [3], or variations thereof. In GitHub, this is simplified, with issues being either *open* or *closed*. This simplified representation highlights a gap in demonstrating the complete life cycle of an issue. As mentioned earlier, an issue can report a defect, an improvement, provide help to users, or other types of reports. In essence, when an issue reports a bug, it may involve a defect, and the defect life cycle should encompass stages such as insertion, detection, and removal [43].

The defect detection process, as depicted in the defect life cycle, is crucial for understanding the defect and its impact, which is the first step towards fixing it. Typically, users carry out the detection phase when reporting issues in repositories that have issue trackers [28]. Subsequently, these contributors are tasked with resolving these issues either by attempting to reproduce the defect themselves or by assigning it to other contributors responsible for addressing the issue [40].

When managing the defect resolution process, triage plays a crucial role in investigating, characterizing, prioritizing, and assigning issues for correction [4, 26]. Therefore, reports typically include fields to classify severity and versioning [14]. Not all issues are equally severe or have the same priority in the issue tracker. Contributors commonly prioritize issues based on factors such as defect priority, severity, time to resolution, product importance, developer availability, report quality, and effort required [26, 41]. The detection process conducted during triage is also critical for identifying and addressing bugs, and defect classification helps quantify these issues [38].

The triage process involves analyzing all defect information contained in the issue by the responsible party, and this process can be time-consuming [25]. Developers carefully consider the factors reported in the issues to avoid misclassifications during this selection stage. However, incorrect bug classifications still occur and can potentially harm the software [21].

As highlighted by Kaushik et al. [26], the defect management process progresses from triage to resolution and ultimately validation of the defect resolution. The time taken to fix a bug is a

[1] https://figshare.com/s/efba88b6b88a013056b2

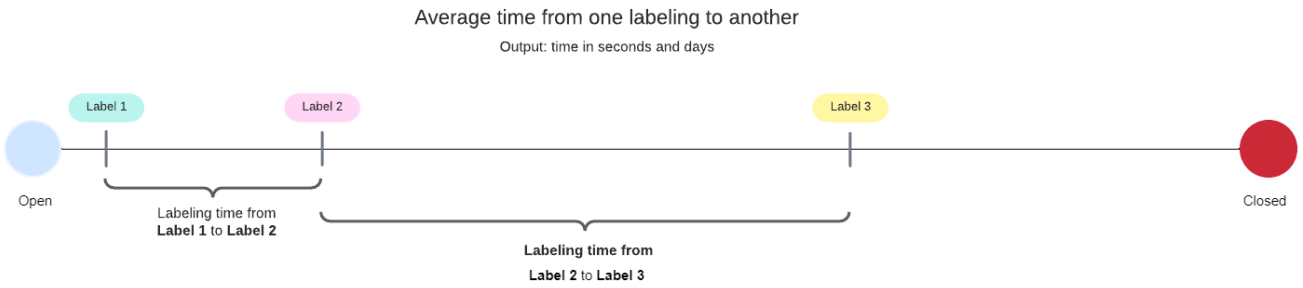

Figure 1: Illustration of the issue triage time considered in RQ2, showing the time interval between Label 1 and Label 2, indicating the average time for labeling issues in different phases of issue resolution.

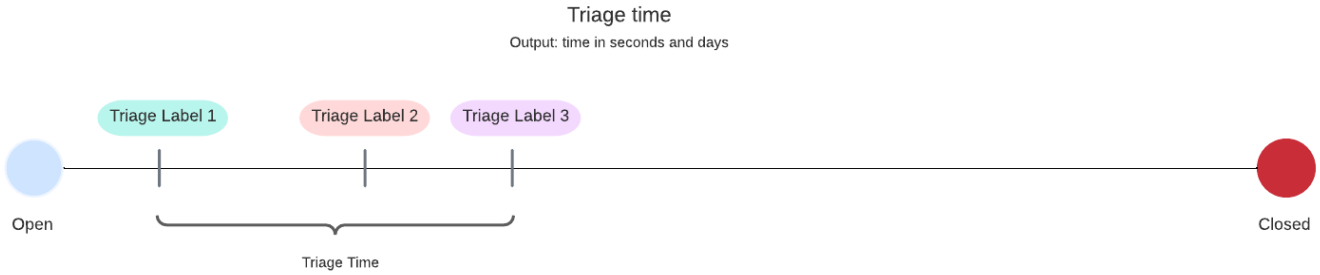

Figure 2: Illustration of the issue triage time considered in RQ3, depicting the time from issue opening to the last triage-related label (Triage Label 3), characterizing the triage phase duration for the issue.

critical factor that reflects the analysis, complexity, and priority of the issue [27]. Such analyses can provide valuable insights for the triage process, helping filter and replicate successful aspects and classifications that proved effective in resolving the issue.

The standard fields of a GitHub issue adhere to the issue tracker model, comprising the title, description, issue insertion date, author, and status (open or closed). This structure is prevalent across various platforms, such as Bugzilla[2] and Jira[3]. It is crucial to fill these fields comprehensively and accurately to aid in issue resolution. GitHub supports issue reporting using the Markdown markup language [18], allowing for extensive text and formatting options. This functionality empowers developers to effectively address reported issues [4, 14]. Conversely, inaccuracies, errors, or an inability to reproduce reported defects can significantly delay the entire defect resolution cycle [25].

The GitHub issue-reporting also includes tracking all activity performed (assignments [17], comments, opening, closing notifications, and others) from the issue's event list. In addition, it enables the creation of custom labels and attributing multiple labels to a single issue.

2.2 Issue labeling

Labeling is present in the software development process, which can provide facilities for those on the other side performing this difficult task and can be helpful in many software artifacts such as architecture, components, documentation, and testing [37]. They are concepts in social community systems related to free writing personalization of information of these systems regarding any function, including reuse, management, and re-finding information[36].

Nevertheless, considering the context of reporting issues, this concept about the labels being a free field for the contributors to build a customizable environment for each report is also valid on GitHub repositories. The contributors could create new labels, assign labels, or insert information into labels, meaning adding more information to solve the issue.

There are two ways to use the issue labeling tool: using GitHub suggestion labels or creating custom labels free of charge. We call the first option standard labels and the second option custom labels the user creates.

- **Standard labels:** GitHub provides a list of nine labels [4] when the developers create the repository. They are: *bug, documentation, duplicate, enhancement, good first issue, help wanted, invalid, question* and *wontfix*.

[2]https://www.bugzilla.org/
[3]https://www.atlassian.com/br/software/jira

[4]Available at https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/managing-labels#about-default-labels

- **Custom labels:** GitHub also provides a tool for creating and editing labels so that the developer can develop customizable labels for their repository. Each new label should include a name (preferably representative) and a brief description. The developer could also associate a *color* to the label. This process involves writing the label's name, a short description, and choosing the color the label will represent.

3 Research strategies and tools

We start by describing the study design and how data from open-source repositories was selected, mined, and processed. Finally, we explain the research questions and how our methods in data processing led to obtaining the data for this study's results.

3.1 Study Design

The study design followed four phases, which were fundamental for obtaining, processing, and finally answering the research questions. Figure 3 provides an overview of the study design workflow. The 4-phase design encompasses *selection, mining, pre-processing,* and *data processing*. Next, we proceed with reporting the findings. These are detailed next.

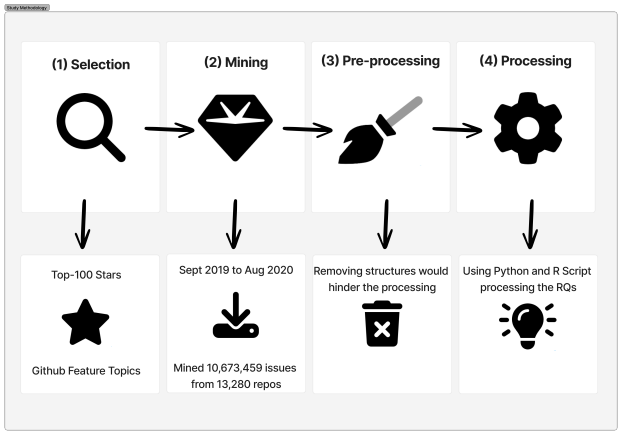

Figure 3: Overview of the study design workflow.

(1) Repository Selection Criteria: To meet our characterization goals, we carefully selected a set of relevant repositories for our study. The aim was to create a diverse issue dataset covering various families and domains. Our selection criteria stipulated that all chosen repositories must be hosted on GitHub and align with the list of featured topics provided on the platform [16]. This list encompasses 180 featured topics on GitHub, offering a wide array of system repositories across different domains and programming languages. Since each topic includes numerous systems, we needed a metric for selecting repositories. Given that the star count is a popular metric for measuring repository popularity on GitHub [10], we opted to select the 100 most popular repositories for each topic based on their star count.

(2) Mining Repository Issues: Following the selection of repositories for each topic, we initiated the mining process. This involved collecting data related to issues and pull requests. Specifically, we gathered metadata such as *id, author, title, body, description, status, creation date, events, repository labels, issue labels, reactions*, and *comments* using the GrumPy tool [22]. As an additional step, we included repository labels as an extra field in our database of issues to streamline our analysis. The mining process yielded a dataset comprising 10,673,459 issues from 13,280 repositories belonging to GitHub's featured topics (Supplementary Material), spanning the period from September 2019 to August 2020.

(3) Pre-processing Issues' Components: An issue contains a wealth of information, but not all of it is relevant to our study. The pre-processing phase is crucial for removing unnecessary details that could potentially alter the characteristics of our results. Certain structures need to be eliminated to streamline further processing of issue information, especially since GitHub issues' components are formatted using Markdown [18]. During the pre-processing phase, we removed the following structures from the issues: *code quotes, tags, hyperlinks, tables and references, citations and repeated comments, warnings, exceptions, class names, paths, special characters, numbers, multiple blank spaces,* and *stop words* from the title, body, and comments. This cleaning process ensures that our analysis focuses on the most relevant and meaningful information from the issues. Next, we explain each of these steps in more detail:

- Tokenization: The summary was divided into smaller units called tokens, words, syllables, or letters. Tokenization by word was used in this study;
- Normalization: The tokens were converted to lowercase to achieve standardization and avoid duplication of words due to case sensitivity. Symbols, punctuation marks, line breaks, emojis[5], and non-ASCII characters were removed, considering only alphanumeric characters. This step is really important to reduce noise and ensuring consistency in the data;
- Removal of stopwords: Words that do not provide significant discriminatory information, known as stopwords (e.g., articles, prepositions), were removed from the tokens. These words often appear with considerable frequency and do not contribute to the training process;
- Lemmatization: The tokens were grouped according to their lemma, which is their canonical form. For example, plural forms were reduced to their singular form. Lemmatization helps in consolidating words with similar meanings and reducing the dimensionality of the data;
- Stemming: The tokens were reduced to their base or dictionary form, removing affixes such as prefixes and suffixes. This process maps words to their root form, allowing for better performance and grouping of words with similar meanings.

(4) Processing Phase:

We implemented another Python script to analyze the issues in the database and process the labels' occurrences to address the stated RQs. The steps to gather and analyze data are described next.

(1) **Answering RQ1:** To address RQ1, we should search and count the creation of custom labels and usage of standard labels by the repositories in the issue using a counting algorithm. The algorithm analyzed each issue with the count

[5]Ideograms and smileys usually used in informal texts

and characterization of standard and custom labels, so they were processed differently, with each group counted. The algorithm verifies if repository issues used all the created custom labels. RQ1 also involves studying the standard labels made available by GitHub, which analyzes these labels' usage by contributors. The data collection to calculate this RQ involved using the most used labels in each repository and the entire database. The processing consists of detecting the use of labels by checking all labels and the most used by each repository.

(2) **Answering RQ2 and RQ3:** To answer RQ2 and RQ3, we identified labels from the triage and defect detection process with manual classification identifying labels in a list of the most used labels in the issues. We manually interpreted and studied those most related to the triage and detection processes and found many repeated custom labels written that had the same meaning. After recognizing these labels, we use a Python algorithm to go through the time events that occurred in the issue and search for occurrences of labels that have previously classified labels. Thus, we extract the time of the Github actions history and make time calculations for each action that occurred in the issue. Thus, when a label that has been cataloged is found, we capture the time of the action and calculate until the occurrence of a label from another development phase. For example, we found a label at time t1 related to triage, so we started searching forward and found another label at time t2 related to the development phase. Hence, the issue's triage phase went from creation until the labeling occurred in time t1. Just as the development phase starts at label time t2 until another label about the change validation phase appears in the issue.

The results of the algorithm processing the dataset and counting the data from **RQ1**, **RQ2**, **RQ3**, generated four tables about the results of the 13,280 repositories. We developed an R script to analyze, calculate, and create the necessary graphs to present the results from the results table extracted from the database. All of these materials, including the tables and the R script, are available at the supplementary material repository.

4 Results

RQ1: How often do developers use labels in open-source repositories?

Given both sets of labels ("standard" and "custom"), we start by showing the results of the first. We found 9,713 (73.14%) repositories using the standard labels, while 3,567 (26.85%) repositories have not used any standard labels. Figure 4 shows the use standard labels.

The standard labels `bug`, `enhancement`, and `question` are the most used in the labeling activities. This result shows the use of standard labels to identify the type of issue to which it refers. Thus, one applicability of those labels given by the practitioners is the monitoring and quick identification of the content type of these issues.

By analyzing the use of custom labels, we found that 4,703 (35.41%) repositories do not create any custom label and 8,577 (64.58%) repositories created 149,122, with a median of 2 labels

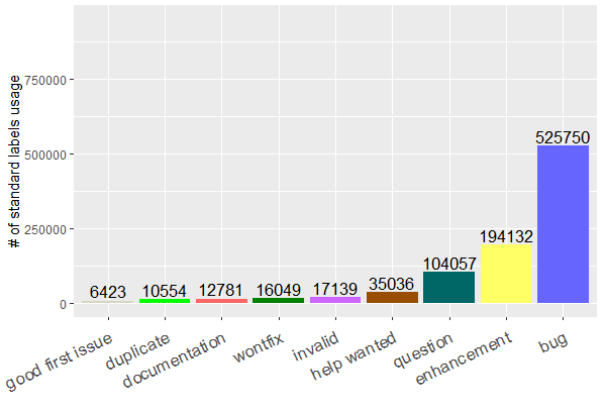

Figure 4: Standard labels and usage

at least to the repository. Figure 5 shows the top-30 custom labels we found in this analysis.

Since an issue could be labeled with one or more labels, we found that custom labels were used 9,322,003 times in the database for label issues, with a mean of 701.95 issues and a median of 6 labels.

The spearman's correlation coefficient was found with p-value of 0.752. It indicates a significant correlation between the number of issues and the number of labels created.

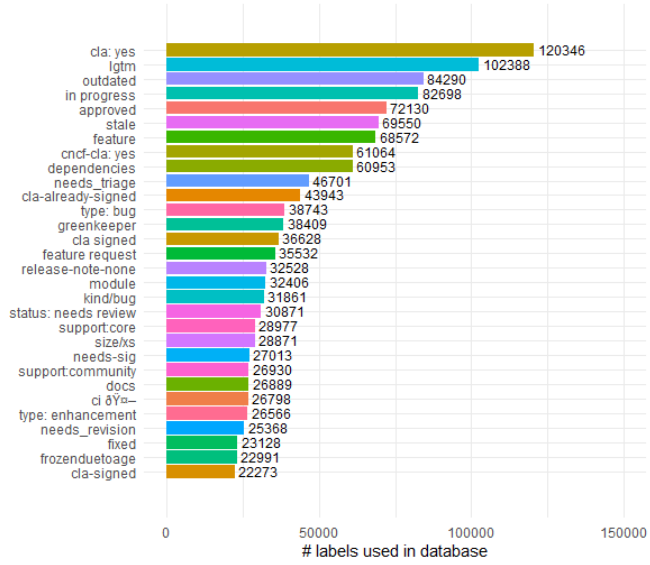

Figure 5: Top-30 custom labels.

We calculate the correlation between the number of issues and custom label usage to determine a relationship between them. The spearman's correlation coefficient was found with p-value of 0.785. It suggests a significant correlation between the number of issues in a repository and the use of custom labels for labeling. Figure 6 plots the number of issues and the custom labels usage.

The results show that over half of the repositories create new labels, but not all use them in the labeling practice. Additionally, we

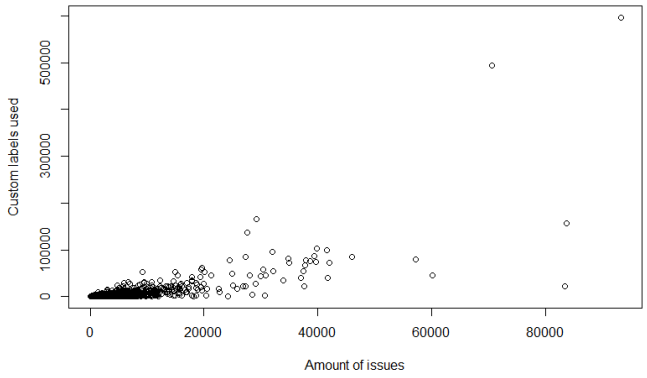

Figure 6: Plot of number of issues and number of custom labels used

observed a correlation between the number of issues in a repository and the number of custom labels. These results show that the more issues a repository has, the more it creates and uses custom labels.

We investigated the custom labels and their importance in the triage and defect detection. A manual classification was performed in the study and showed that the first 1,752 labels of that list were classified. We removed labels without meaning for the issue resolution process from the list. Figure 7 shows a treemap chart detailing the most used labels in each category with the number of occurrences. We identified labels with different purposes, such as requesting revision, prioritization, severity assigning issues, identifying issues, and finding/classifying defects.

In the defect investigation phase, three of the labels that identify this purpose were *needs_triage*, *needs_revision*, *pending discussion*. The prioritization and severity classification contrasted with labels indicating the priority using the low, medium, and high priority scales in label issues. Some issues also present labels with a priority request with the label needs-priority. Labels were found to identify an issue's type, including defect classification, improvement, support, and bug report. The contributors use labels to indicate additional information about the reported defect in areas about modules and packages. The creation of labels also covered the implementation and validation phases. Some labels were related to the commit, build resolution progress information, code refactoring, and indications in the merge processes. The frequency of labels related to validating the issue resolution code was significant. Those labels show information about the validation process, specifically when the process is pending, carried out, or finished. We highlight some evident labels in their use in the triage phases involving prioritization, assignment, investigation, location, and classification of defects and types of issues.

The implementation and validation part also reveals that contributors use this purpose. Labels are perceived to be applicable for different purposes in a project. Still, our study shows contributors use this for triage and identification in the solution's implementation and validation phases. Even in smaller quantities and not being a frequently used practice, repositories use this labeling tool for organization and control from defined phases of issues.

RQ2: What is the average time developers take to label issues during the issue resolution?

When analyzing the labeling time between labels, we found an average number of 114.3 days, with a median of 17 days and a standard deviation of 237.5. The time between labels ranges from a minimum of one day to a maximum of 2664 days.

Figure 8 shows the labeling frequency in days. Most labeling activity is concentrated before day 100. The interval between one label and another varies much over time. Times a peak of labeling between the first few days, so changing, adding, or replacing a label is performed in less time. Two probable explanations, one being that multiple labeling coincides, and the other is that the label change follows a predetermined period defined in the repository. This defined period may characterize an issue phase, i.e., triage, implementation phase, or validation of the changes. In Figure 8, the data is more concentrated at the bottom of the graph, indicating that labeling occurs in less time.

Figure 9 shows a set of histograms representing 5, 20, 100, and 500-day time interval. In all representations, we could observe a concentration of the labeling activity on the left side.

RQ3: Can we define the triage phase based on labeling practices?

After analyzing the labeling time between one label and another, we have a pattern where labeling is performed quickly. Thus, after selecting labels ranked by their occurrence in the resolution process, we have a set of sorting labels selected. Thus, going through this database, we find what can be said to be the triage period of an issue from the use of triage labels. The triage time mean is 240.3 days, the median is 30, and the standard deviation is 551.9. Moreover, the time spent in triage ranges from one day to a maximum of 7029 days. According to the data, the majority of issues were triaged between 1 and 100 days, as Figure 10 shows.

We also divided the analysis into days 5, 100, and 500, and 2000 of triage labeling. Figure 11 shows the set of histograms, and Figure 12 shows the set of Violin Plots for such interval days. They indicate that the labeling time is concentrated in a shorter time of days and that the vast majority of the triage performed by the labels takes little time, from 1 to 100 days, to complete.

5 Discussion

With this feature topics repositories analysis, we could characterize some trends in standard and custom label usage. We have seen that the standard GitHub labels are widely used and that only a small number of repositories do not use them. The three leading most used standard labels identify the issue's characteristics, such as bug, improvement, and help. This result helps to reinforce and complement the role of labels to inform the issues to assist in the identification of the type of the issue. The label creation and customization tools are also widely used by repositories that also customize and use labels for the purposes required in their repository issues — there is a significant correlation between the repositories with more issues and the creation of more labels. Therefore, the repositories

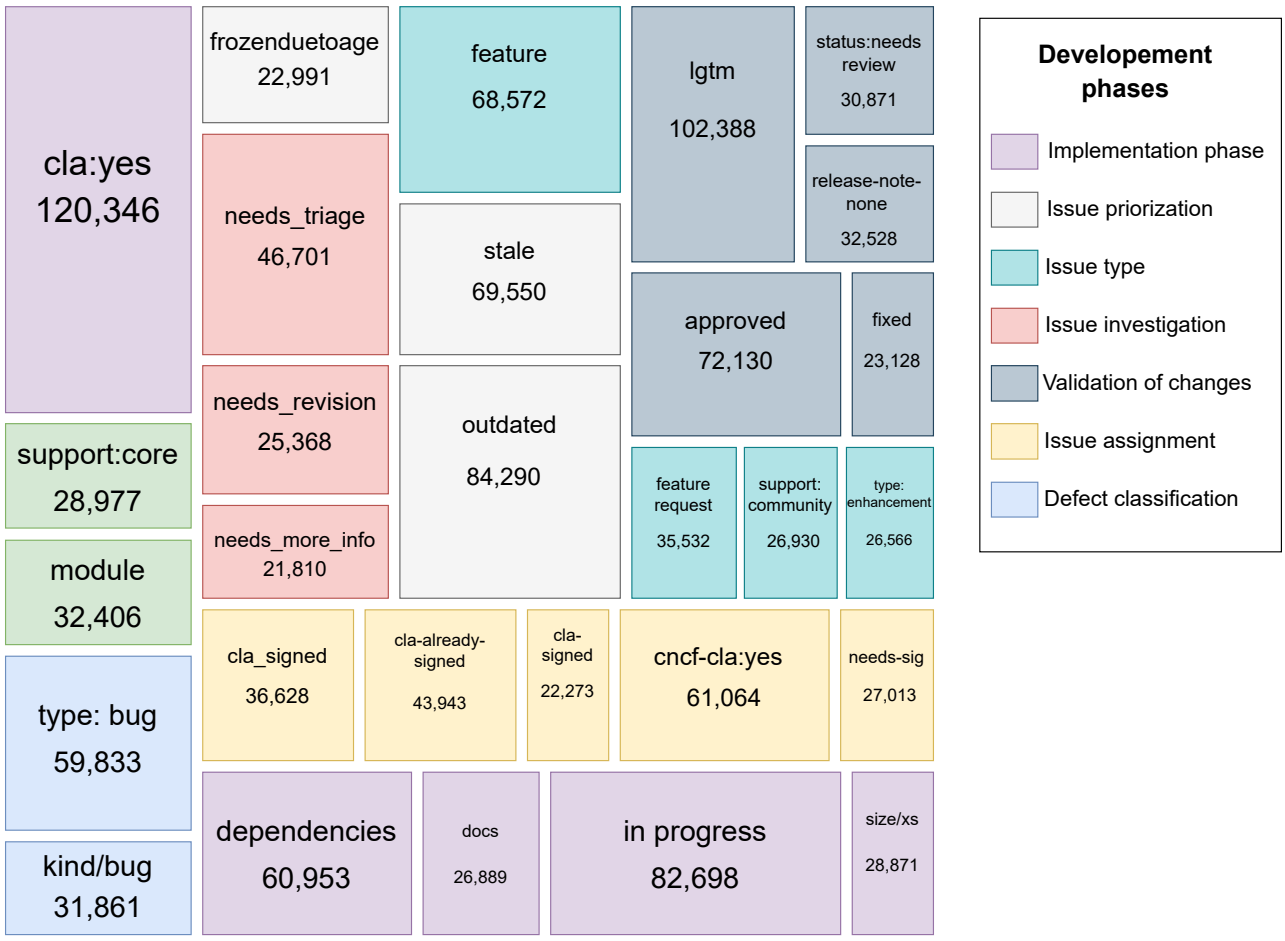

Figure 7: Treemap chart detailing the most used labels in each category with the number of occurrences.

need to create more labels to satisfy the classification needs of the reported issues. Furthermore, the bigger this repository the more it needs labels to manage its reports better, thus fulfilling the defined process.

A large majority of labels are used for the implementation phase, change validations. The triage phase includes defect localization, issue prioritization, issue type, issue investigation, issue assignment, and defect classification. We have provided evidence of customized labels to enhance the triage, cycle, and defect detection processes. Our findings demonstrate that utilizing these structures in such a manner serves crucial roles in organizing and maintaining the repository effectively. Labels are not only social elements but also technical components that shape the organization of open-source repositories by facilitating the identification, implementation, and validation of code changes.

The implementation of these labels aids project developers in triage, cycle management, and defect detection, showcasing their potency in establishing contributor support and bolstering control, identification, and maintenance mechanisms for the repository. These concepts are deeply rooted in Software Engineering principles, showcasing the full spectrum of support and adaptability offered by labels in repository management.

6 Threats to Validity

6.1 Internal Validity

The most evaluated repositories in the list of topics can bring more repositories from a particular area. The list of repositories does not have the same characteristics, or it is the same work area. Still, when mining these topics, we consider 100 repositories at each list to homogenize the database with popular repositories. The study only handled repositories that contained issues. Thus, repositories that did not contain issues were not included by the algorithm in the results table because they do not have issues to be analyzed.

6.2 External Validity

By only using GitHub repositories, the generalization of the study to other platforms may not present the results obtained in this study. In this study, we focused on using GitHub topics with a diversity of

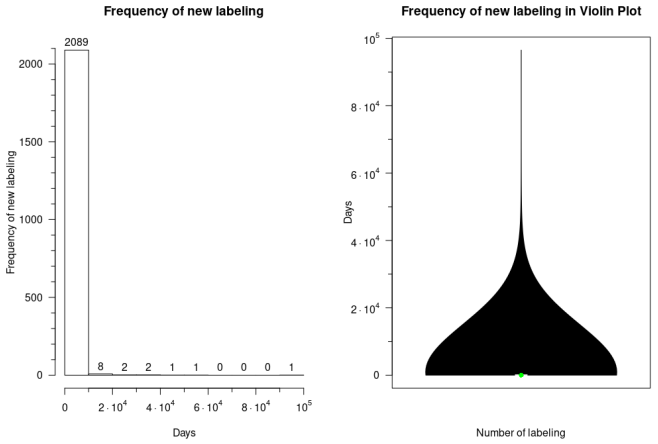

Figure 8: Histogram and Violin Plot of labeling frequency versus days with no outliers

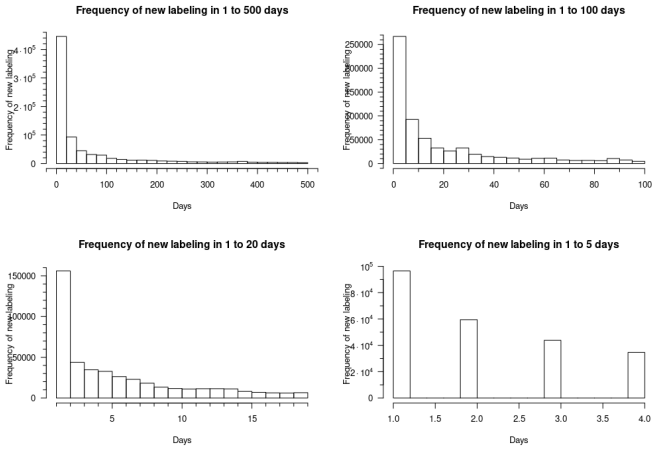

Figure 9: Histogram of 5, 20, 100, and 500-day time interval.

repositories from different domains. This selection of repositories decreases the bias in the analysis of our study. We mined the repositories used in this study in a specific time frame, and the issues reported, later on, were not considered in this analysis.

6.3 Construct Validity

The choice of metrics and graphs may not be well suitable to represent the data. We performed the search and chose different representations to avoid bias and inform about label and usage data in the repositories. From the repositories' selection with the popularity metric to the analysis of correlation, labeling time, and the numbers related to the labels themselves, we always aim to present the data in the best possible way and with no interference in the results.

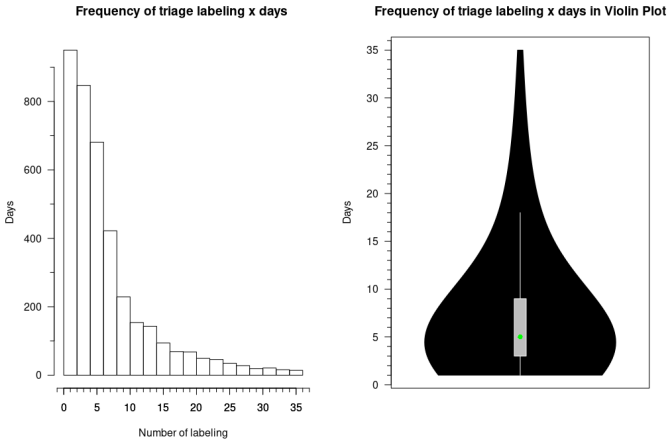

Figure 10: Histogram and Violin boxplot of triage labeling frequency versus days with no outliers

7 Related Work

Extensive research has been done by researchers to understand the behavior of issues in open-source issue trackers. The study conducted by Treude and Storey [37] obtained significant results in tags usage in software development regarding a social and technical context. It also defined the concept of labels for Software Engineering in an industrial development environment using the Jazz project life cycle management platform. The way pursued by our study is related to the application of labels in a technical course in open-source systems. We analyze the creation and use of standard and customizable labels and their trends in various repositories with different domains and practices in a more current context, open-source systems. The focus also involved the technical occurrence of the issue labels and the emphasis on triage for defects in open-source repositories.

Bissyandé et al. [9] present a comprehensive investigation to understand issues and the repository actions in GitHub, including a short overview of labels used in reported issues classified as issue type: bug report and feature request. Our study went deeper into the analysis of the labels, identifying which GitHub provides labels and which are created by users and the creation trend. Our study performed a correlation between the number of issues and the creation of labels, resulting in a significant correlation between the two actions. Our analysis focused on the labels in the triage process and what issues contributors are based on when they are in the resolution process.

Cabot et al. [11] investigated the application of issues labels on GitHub repositories, describing the number of labels, the most used ones, and their influence on the project through a correlation between time to solve and issue age grouping types of labels. On the other hand, our study focuses further on creating custom labels and finds a correlation between the creation and the number of issues and time of each issue phase.

Using predictive systems, Alonso-Abad et al. [2] investigate a system that labels issues based on machine learning to reproduce the labels on new issues. This work also presents data on issue

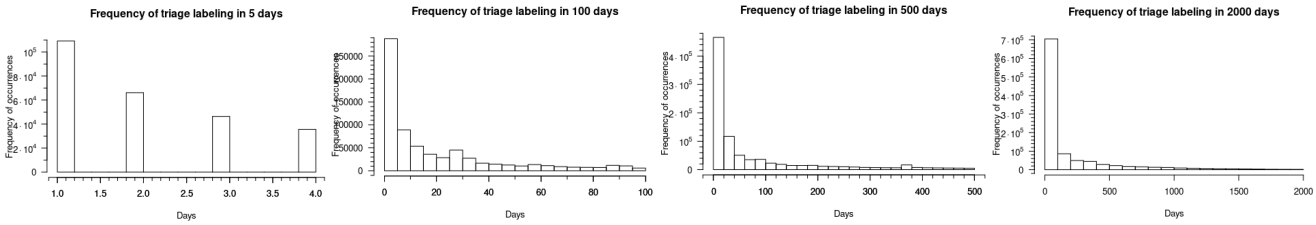

Figure 11: Histogram of triage labeling frequency in 5, 100, 500, and 2000.

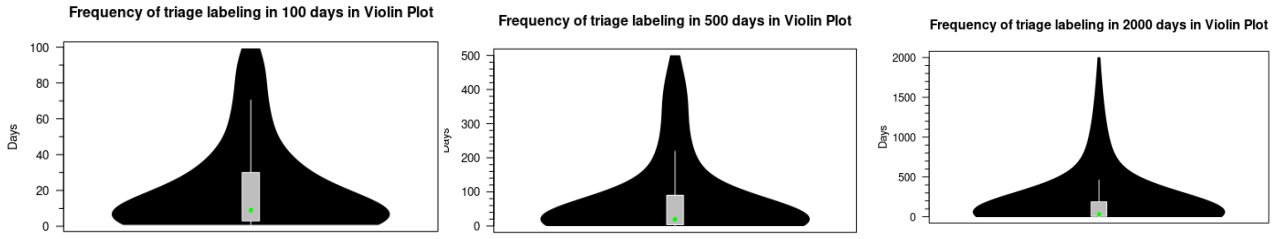

Figure 12: Violin Plot of triage labeling frequency in 5, 100, 500, and 2000.

labels and a list of labels most used by these repositories. There is an analysis of the algorithm's performance presented with a focus on the best performance to be used to label the repositories. Our work focuses only on the analysis of data from the repositories and separating the types of labels as standard and customizable, with a focus on the analysis of issue developer time and identifying their phases in the issue development cycle.

8 Conclusions

In this study, we aimed to examine GitHub repositories with higher ratings and analyze their issues, encompassing a total of 13,280 repositories.

To accomplish the goals, we employed various data manipulation techniques, including selection, mining, pre-processing, processing, and reporting findings. The study revealed that 73.14% of the analyzed issues utilized labeling, while 26.85% did not. We found that 35.41% of labeled issues did not involve custom labels, while 64.58% utilized a median of 2 labels. Besides, 65.59% of labeled issues with the median usage were labeled by professionals. Custom labels were applied a significant 9,322,003 times, highlighting their importance.

These findings reveal that, while issue labeling is not universally adopted, labeled issues are more easily discernible, particularly in large repositories where numerous issues exist. Labeling also aids in bug detection and the triage process, functioning as a filter or keyword for identifying issues or their states.

The frequency of labeling shows significant temporal variations, which may be tied to concurrent labeling activities or predetermined periods within the repository. In the latter scenario, issue categorization may pertain to triage, implementation phases, or validation of changes.

Indeed, we could identify several gaps to discuss and investigate. Particularly, regarding the creation and utilization of multi-labels in issues. Additionally, there is a need for guidelines to assist contributors in creating and using labels effectively within repositories.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001; CNPq grants 315840/2023-4 and 403361/2023-0; and FAPESB grant PIE0002/2022.

Artifact Availability

The research package, including the issues database, results tables, and scripts utilized in this study, is accessible in the supplementary material repository at https://figshare.com/s/efba88b6b88a013056b2.

References

[1] 2021. *Modern Software Engineering: Doing What Works to Build Better Software Faster* (1st ed.). Addison-Wesley Professional.
[2] Jesús M Alonso-Abad, Carlos López-Nozal, Jesús M Maudes-Raedo, and Raúl Marticorena-Sánchez. 2019. Label prediction on issue tracking systems using text mining. *Progress in Artificial Intelligence* 8, 3 (2019), 325–342.
[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an Open Bug Repository. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange* (San Diego, California) *(eclipse '05)*. ACM, New York, NY, USA, 35–39.
[4] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who Should Fix This Bug?. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. ACM, New York, NY, USA, 361–370.
[5] J. Aranda and G. Venolia. 2009. The secret life of bugs: Going past the errors and omissions in software repositories. In *2009 IEEE 31st International Conference on Software Engineering*. 298–308.
[6] Mario Luca Bernardi, Gerardo Canfora, Giuseppe A. Di Lucca, Massimiliano Di Penta, and Damiano Distante. 2012. Do Developers Introduce Bugs When They Do Not Communicate? The Case of Eclipse and Mozilla. In *2012 16th European Conference on Software Maintenance and Reengineering (CSMR '12)*. IEEE Computer Society, USA, 139–148.
[7] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiß, Rahul Premraj, and Thomas Zimmermann. 2007. Quality of Bug Reports in Eclipse. In *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology EXchange* (Montreal, Quebec, Canada) *(eclipse '07)*. ACM, New York, NY, USA, 21–25.

[8] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What Makes a Good Bug Report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Atlanta, Georgia) *(SIGSOFT '08/FSE-16)*. ACM, New York, NY, USA, 308–318.

[9] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 188–197.

[10] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software* 146 (2018), 112 – 129.

[11] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. 2015. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 550–554.

[12] Yguaratã Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, Tassio Ferreira Vale, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. 2014. Challenges and opportunities for software change request repositories: a systematic mapping study. *J. Softw. Evol. Process.* 26, 7 (2014), 620–653.

[13] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM.

[14] Steven Davies and Marc Roper. 2014. What's in a Bug Report?. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Torino, Italy) *(ESEM '14)*. ACM, New York, NY, USA.

[15] Andrea Di Sorbo, Giovanni Grano, Corrado Aaron Visaggio, and Sebastiano Panichella. 2021. Investigating the criticality of user-reported issues through their relations with app rating. *Journal of Software: Evolution and Process* 33, 3 (2021), e2316. https://doi.org/10.1002/smr.2316 e2316 smr.2316.

[16] Github Inc. 2020. Page of all features topics in Github. https://github.com/topics

[17] Github Inc. 2024. Assigning issues and pull requests to other GitHub users. https://docs.github.com/pt/issues/tracking-your-work-with-issues/assigning-issues-and-pull-requests-to-other-github-users

[18] Github Inc. 2024. Basic writing and formatting syntax. https://guides.github.com/features/mastering-markdown

[19] Yingying He, Wenhua Yang, Minxue Pan, Yasir Hussain, and Yu Zhou. 2023. Understanding and Enhancing Issue Prioritization in GitHub. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 813–824. https://doi.org/10.1109/ASE56229.2023.00044

[20] Jueun Heo and Seonah Lee. 2023. An Empirical Study on the Performance of Individual Issue Label Prediction. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. 228–233. https://doi.org/10.1109/MSR59073.2023.00041

[21] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *35th International Conference on Software Engineering (ICSE)*. 392–401.

[22] Joselito Mota Jr., Railana Santana, and Ivan Machado. 2021. GrumPy: an automated approach to simplify issue data analysis for newcomers. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering* (Joinville, Brazil) *(SBES '21)*. ACM, 33–38. https://doi.org/10.1145/3474624.3476012

[23] Joselito Júnior, Gláucya Boechat, and Ivan Machado. 2021. Label it be! A large-scale study of issue labeling in modern opensource repositories. In *24th Iberoamerican Conference on Software Engineering (CIbSE 2021)*. Curran Associates, 262–275. https://arxiv.org/abs/2110.01328

[24] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 406–409. https://doi.org/10.1109/ICSME.2019.00070

[25] Jaweria Kanwal and Onaiza Maqbool. 2012. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology* 27, 2 (2012), 397–412.

[26] Nilam Kaushik, Mehdi Amoui, Ladan Tahvildari, Weining Liu, and Shimin Li. 2013. Defect Prioritization in the Software Industry: Challenges and Opportunities. In *IEEE Sixth International Conference on Software Testing, Verification and Validation*. 70–73.

[27] Sunghun Kim and E. James Whitehead. 2006. How Long Did It Take to Fix Bugs?. In *Proceedings of the 2006 International Workshop on Mining Software Repositories* (Shanghai, China) *(MSR '06)*. ACM, New York, NY, USA, 173–174.

[28] Ran Mo, Shaozhi Wei, Qiong Feng, and Zengyang Li. 2022. An exploratory study of bug prediction at the method level. *Information and Software Technology* 144 (2022), 106794. https://doi.org/10.1016/j.infsof.2021.106794

[29] Audris Mockus. 2010. Organizational Volatility and Its Effects on Software Defects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Santa Fe, New Mexico, USA) *(FSE '10)*. ACM, New York, NY, USA, 117–126.

[30] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating github for engineered software projects. *Empirical Software Engineering* 22, 6 (2017), 3219–3253.

[31] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. 2002. Evolution Patterns of Open-Source Software Systems and Communities. In *Proceedings of the International Workshop on Principles of Software Evolution* (Orlando, Florida) *(IWPSE '02)*. ACM, New York, NY, USA, 76–85.

[32] Maleknaz Nayebi, Shaikh Jeeshan Kabeer, Guenther Ruhe, Chris Carlson, and Francis Chew. 2018. Hybrid Labels Are the New Measure! *IEEE Software* 35, 1 (2018), 54–57.

[33] Daniel Rodriguez, Israel Herraiz, and Rachel Harrison. 2012. On software engineering repositories and their open problems. In *First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*. 52–56.

[34] Kurt Schneider and Jan-Peter von Hunnius. 2003. Effective experience repositories for software engineering. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE.

[35] Mohammed Latif Siddiq and Joanna C. S. Santos. 2022. BERT-Based GitHub Issue Report Classification. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. 33–36. https://doi.org/10.1145/3528588.3528660

[36] Margaret-Anne Storey, Jody Ryall, Janice Singer, Del Myers, Li-Te Cheng, and Michael Muller. 2009. How Software Developers Use Tagging to Support Reminding and Refinding. *IEEE Transactions on Software Engineering* 35, 4 (July 2009), 470–483.

[37] Christoph Treude and Margaret-Anne Storey. 2012. Work Item Tagging: Communicating Concerns in Collaborative Software Development. *IEEE Transactions on Software Engineering* 38, 1 (2012), 19–34.

[38] J. H. van Moll, J. C. Jacobs, B. Freimut, and J. J. M. Trienekens. 2002. The importance of life cycle modeling to defect detection and prevention. In *10th International Workshop on Software Technology and Engineering Practice (STEP '02)*. IEEE Computer Society, 144–155.

[39] Jun Wang, Xiaofang Zhang, Lin Chen, and Xiaoyuan Xie. 2022. Personalizing label prediction for GitHub issues. *Information and Software Technology* 145 (2022), 106845. https://doi.org/10.1016/j.infsof.2022.106845

[40] Wenxin Xiao, Jingyue Li, Hao He, Ruiqiao Qiu, and Minghui Zhou. 2023. Personalized First Issue Recommender for Newcomers in Open Source Projects. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 800–812. https://doi.org/10.1109/ASE56229.2023.00158

[41] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In *34th International Conference on Software Engineering (ICSE)*. 25–35.

[42] Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. 2019. Mining Software Defects: Should We Consider Affected Releases?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 654–665. https://doi.org/10.1109/ICSE.2019.00075

[43] D Zubrow. 2009. IEEE Standard Classification for Software Anomalies. *IEEE Computer Society* (2009). IEEE 1044-2009.