Understanding Feature Request Practice on GitHub via a Large-Scale Empirical Study

Jiajun Li[†], Wenhua Yang[†§*], Minxue Pan[§], Yu Zhou[†]
[†]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
[§]State Key Laboratory for Novel Software Technology, Nanjing University, China
Email: {lijiajun, ywh}@nuaa.edu.cn, mxp@nju.edu.cn, zhouyu@nuaa.edu.cn

***Abstract*—Feature requests are a key communication mechanism on GitHub, enabling users and developers to collaboratively shape the direction of open-source projects. Feature requests are prevalent and important, but have been underexplored in existing studies. There is limited understanding of how they are labeled, how they evolve, and how they are resolved. A deeper understanding of feature requests is critical, not only for improving issue triage and project management but also for fostering more effective collaboration within open-source communities. In this work, we present the first systematic and large-scale empirical study of feature requests. Drawing on 1.4 million issues from 825 GitHub repositories, we examine how feature requests are labeled, how their submission and backlog patterns change over a project's lifecycle, how they differ from other types of issues in terms of resolution and engagement, and what factors contribute to their successful handling. Our findings reveal that labeling practices are often inconsistent across projects, that feature requests follow distinct temporal trends, and that those which are lengthy and contain large code snippets tend to be more difficult to resolve. By contrast, concise and clearly defined requests, particularly those submitted by experienced contributors and accompanied by active discussions, are more likely to be addressed. This study underscores the challenges of managing feature requests at scale and provides practical insights for maintainers, contributors, and researchers. To support future work in this area, we publicly release our dataset and analysis results.**

***Index Terms*—GitHub, feature requests, issue tracking**

I. Introduction

GitHub is a widely used platform for open-source collaboration and supports key aspects of modern software engineering. Its support for pull requests, branching, and issue tracking facilitates distributed development and requirement management. Among these, issue tracking is especially important, serving as both a project management tool and a communication channel. GitHub's issue tracking system provides several default labels, such as *bug*, *documentation*, and *good first issue* [1]. Among these, the *enhancement* label is commonly used to indicate requests for new features. Following established definitions in prior work [2]–[4], we define feature requests as issues that propose new functionalities or improvements to existing ones.

Feature requests have long played a vital role in software development. Before the advent of collaborative platforms, development teams typically relied on mailing lists, meeting notes, and manually curated documents to track such requests. However, these methods often lacked efficiency, transparency, and scalability. The emergence of internet-based tracking systems such as Bugzilla and Jira marked a significant improvement, introducing more structured and collaborative mechanisms for managing feature requests. Building on these advances, GitHub adopts a more open, community-driven model, where feature requests serve not only as a feedback channel between users and developers but also as a lens into broader trends in software evolution and open-source collaboration. Notably, feature requests are among the most frequently reported types of issues on GitHub [5].

Although feature requests are important, they remain an understudied aspect of GitHub's issue tracking system. Existing research has primarily focused on issue tracking mechanisms and the usage of specific labels [6]–[12]. For example, Bissyande et al. [6] examined the relationship between issue reporting and project success, while Liu et al. [8] analyzed the impact of pull requests on social influence and bug-fixing efficiency. Other studies have explored issue templates [9], the adoption and resolution of the *good first issue* label [10], and the role of multi-labeling in issue management [11]. However, the use and treatment of feature requests remain largely unexplored. Key questions remain unanswered, such as whether developers consistently adopt the enhancement label, how feature requests evolve over a project's lifecycle, how their resolution differs from other issue types, and what factors influence their handling in open-source projects. Empirical research on feature requests can provide valuable insights into software evolution, developers' decision-making, and requirement management within open-source communities.

To bridge this gap, we leverage large-scale GitHub data to conduct a systematic and comprehensive quantitative analysis of feature requests in issue tracking, examining their usage patterns, lifecycle evolution, and influencing factors. Specifically, we address the following research questions (RQ):

RQ1: How are feature requests labeled and how does their usage evolve over a project's lifecycle? This question examines how feature requests are labeled on GitHub and how they evolve over a project's lifecycle. Although the *enhancement* label is intended for feature requests, its application may vary across projects. We analyze the labels associated with feature requests, identify common and co-occurring labels, and explore labeling practices. We also investigate the temporal dynamics of feature requests by clustering projects based

* Corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00228

on their feature requests' submission and resolution trends, aiming to uncover evolution patterns.

RQ2: How do feature requests differ from other issues in terms of resolution and engagement? Feature requests may exhibit distinct characteristics compared to other issue types, particularly in terms of resolution time, level of engagement (e.g., number of comments), and assignee allocation. By analyzing these aspects, we aim to uncover differences in how feature requests are handled, whether they require more discussion and coordination, and whether their resolution process is more complex than that of regular issues.

RQ3: What factors influence the resolution of feature requests? Various factors may impact the handling of feature requests, including project characteristics, contributor involvement, and community engagement. This question explores how these factors affect the likelihood of resolving feature requests, offering insights into best practices for managing feature-related contributions in open-source development.

Using a dataset of 1.4 million issues from 825 GitHub projects, our study presents the first systematic analysis of feature requests. We found that although the *enhancement* label accounts for nearly 30% of all feature request tags, projects also employ a diverse set of prefixed or alternative labels, which complicates consistent tracking. By clustering submission and backlog trends, we identified three common submission patterns (Stable, Growing, Saturated) and three types of backlog behavior (Responsive, Backlog, Overload). Notably, larger or more popular projects tend to accumulate a growing number of unresolved requests. Feature requests have a median resolution time of 53 days (more than six times longer than regular issues), especially when they are overly verbose or contain excessive code snippets. In contrast, focused requests written by experienced contributors and those that attract active discussion are significantly more likely to be resolved. These findings offer actionable implications for maintainers (to standardize labeling practices and refine triage strategies), submitters (to craft clear and concise feature requests), and researchers (to build predictive tools and explore request dynamics further). All issue data and supplementary materials are publicly available[1] to support future research.

Organization of the paper. Section II outlines our research methodology. Results are presented and analyzed in Section III, followed by implications in Section IV. Threats to validity are discussed in Section V. Related work is reviewed in Section VI, and Section VII concludes the paper.

II. Methodology

In this section, we present the methodology of our study. Figure 1 provides an overview of the research process. We begin by describing how the dataset was constructed from GitHub, followed by a detailed explanation of the analytical approach used to address each research question (RQ1–RQ3). Specifically, RQ1 investigates the labeling and evolution of feature requests, RQ2 analyzes their resolution and engagement in comparison with regular issues, and RQ3 examines the factors influencing feature request resolution. The subsequent subsections elaborate on each step of the methodology in detail.

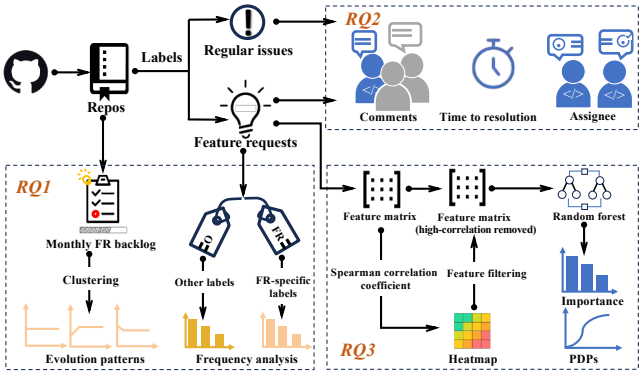


Fig. 1: Overview of the research methodology for this study.

A. Dataset Construction

Selection of repositories. Our study aims to conduct an empirical analysis of feature request issues in GitHub repositories. To ensure a representative dataset, we selected repositories based on specific criteria. Following prior studies [12], we focused on repositories written in the 10 most popular programming languages on GitHub, ensuring that the selected projects primarily represent active software development rather than repositories used for data storage or documentation. To enhance the reliability of our dataset, we applied multiple selection criteria. First, repositories were required to have at least 20 stars to filter out personal or inactive projects while ensuring a minimum level of visibility and community engagement. Second, they needed to have at least one code commit by the end of December 2024 to confirm ongoing development activity. To balance recency and diversity while maintaining a manageable study scope, we randomly sampled 100 repositories per language. Specifically, we first retrieved repositories via the GitHub API using the `--sort updated` parameter to prioritize recently updated projects and exclude outdated or abandoned ones. From this initial set of 2,996 repositories, we then performed random sampling to ensure that the dataset included both highly active repositories and moderately active ones, offering a more comprehensive representation of GitHub's ecosystem. This process yielded an initial dataset of 1,000 repositories.

During our analysis of the initial 1,000 repositories, we found that some did not allow issue submissions. This restriction often came from project-specific management decisions, such as disabling GitHub's issue tracker in favor of external tools or alternative workflows (e.g., discussions, mailing lists, or dedicated project management platforms). Since our study focuses on feature requests within GitHub's issue-tracking system, including such repositories would have introduced

[1]https://doi.org/10.5281/zenodo.15524707

inconsistencies and reduced dataset relevance. Therefore, we excluded them from our analysis. After this refinement, our final dataset included 825 repositories and 1,432,337 issues, focusing on projects with active issue tracking and aligned with our research objectives.

Identification of feature requests. After selecting the repositories for our study, the next step was to identify feature request issues. On GitHub, issues are often categorized using labels, making them a natural starting point for automated identification. Based on GitHub's documentation and our observations across various projects, we initially selected commonly used labels that indicate feature requests, such as "enhancement", "feature request", and "new feature". To account for variations in labeling practices and minimize the risk of missing relevant issues, we manually reviewed the full set of labels used in the selected repositories. This process enabled us to identify additional labels that may denote feature requests. The final set of labels used for identification, comprising 156 unique labels, is provided in our supplementary materials[1]. Using these labels, we extracted all issues tagged with at least one of them as feature request issues. However, since labeling is optional on GitHub, some feature request issues may lack relevant labels. To address this limitation, we applied a secondary filtering step based on textual analysis. Specifically, we reviewed the titles and bodies of issues without feature request-related labels, searching for keywords such as enhancement, feature request, and new feature. Issues containing these keywords were then manually reviewed to confirm their relevance. This process identified only 1,499 additional feature request issues. Ultimately, from the 825 repositories containing 1,432,337 issues, we identified 202,630 feature request issues. In the subsequent analysis, we examine these issues in detail and compare them with other types of issues to gain deeper insights into their characteristics and handling.

B. Investigating Feature Request Labeling and Evolution

In the previous step, we identified feature request issues. Although *enhancement* is the default label on GitHub for representing feature requests, its application varies significantly across projects due to the absence of standardized labeling practices. As a result, feature request issues may be assigned different labels across projects. RQ1 aims to identify the typical labels used for feature request issues and analyze their frequency of usage. To achieve this, we systematically enumerated all labels applied to feature request issues and recorded their occurrences. We then manually classified the labels into two categories: feature request-specific and non-feature request labels. The full classification results are provided in our supplementary materials[1]. This straightforward process was carefully validated through independent checks by multiple authors. It allowed us to analyze the frequency distribution of feature request-specific labels, identify the most prominent ones, and uncover labels that commonly co-occur with feature requests. By analyzing these co-occurring labels, we explored potential labeling patterns and usage practices. This approach provided deeper insights into the logical relationships among labels, offering a more comprehensive understanding of how feature requests are categorized in open-source projects.

In addition, we investigated the evolution patterns of feature requests throughout project lifecycles. This analysis is challenging, as project trajectories can vary widely and temporal patterns are not always clearly discernible. While prior studies on GitHub issues often focus on trends in issue volume over time, few investigate deeper behavioral patterns. Some work has examined issue dynamics, such as creation rates and backlog size, over time [13]–[15], but these studies generally overlook patterns specific to feature requests. To advance this line of work, we conducted a detailed clustering analysis to uncover the evolution of feature requests. We first collected, for each project, the monthly counts of newly submitted feature requests (*new_FR*) and the cumulative counts of unresolved ones (*backlog_FR*). To enable meaningful comparisons, we standardized these time series using z-score normalization.

To ensure reliability, we restricted the observation window to the first 36 months after each repository's creation and excluded projects with shorter lifespans. Of the 825 repositories initially identified, 581 met this criterion and were retained. The 36-month window was chosen to maintain a consistent observation horizon across repositories. Extending the window would reduce the number of projects with complete data and risk biasing the observed trends. In empirical checks with alternative durations, the 36-month cutoff offered a practical balance between capturing temporal dynamics and preserving coverage. For example, extending the window to 48 months would reduce the usable sample from 825 to 499 repositories, a loss of more than 40%. We then applied k-means clustering to group projects based on the temporal trends of the two standardized metrics separately, aiming to reveal distinct evolution patterns. Moreover, we conducted a project-level feature analysis (such as project size, number of contributors, and number of forks) within each cluster to explore potential structural differences and further characterize the observed patterns.

C. Analyzing Feature Request Resolution and Engagement

Feature request issues often reflect users' or developers' expectations for new features and can significantly influence a project's strategic direction. In contrast, regular issues primarily focus on fixing known bugs or making minor feature modifications. These differences may lead to variations in resolution time, engagement levels, and assignee allocation, highlighting the importance of understanding how these issue types are handled differently. To systematically compare feature request issues and regular issues, we examined multiple aspects. First, we analyzed their respective *Time to Resolution* (TTR), which represents the duration from issue creation to closure (measured in days). Only issues with a definite closure time were included, as some issues remained open at the time of data collection. By comparing resolution times, we examined their overall distribution and identified potential outliers for both issue types.

Next, we analyzed engagement levels using the *Number of Comments* as a metric, conducting a comparative analysis between feature request issues and regular issues. Additionally, we considered the assignment of an assignee as another important indicator of resource allocation. To quantify this, we defined the *Assignee Assignment Rate*, which measures the proportion of issues assigned to an assignee within a project. Specifically, for a given project, the assignee assignment rate for feature request issues is calculated as the number of feature request issues with assigned assignees divided by the total number of feature request issues. A similar calculation is performed for regular issues. By treating each project as an independent unit, we compared the assignee assignment rates for both issue types to examine how teams allocate resources. This project-level analysis reveals task distribution patterns across development teams and provides insights for optimizing resource management. To determine whether feature request issues and regular issues exhibit significant differences across these metrics, we applied statistical tests to compare these two groups. This comprehensive analysis offers valuable insights into issue resolution, engagement, and resource allocation, helping teams refine their workflows and improve overall project management.

D. Understanding Factors in Feature Request Resolution

The goal of RQ3 is to examine which characteristics of feature request issues influence their resolution. In earlier steps, we collected a comprehensive set of data related to feature requests, covering aspects such as issue descriptions and the number of comments. Drawing on prior studies [10], [12], [16], we selected a set of features spanning three levels—project, issue, and community (see Table I)—to form a broad and representative basis for analysis. To identify the factors that impact feature request resolution, we adopted a widely used approach in prior work [17]–[19]: building predictive models (e.g., Random Forest) to learn from large-scale data and identify influential features. Before model construction, we conducted a correlation analysis using the Spearman correlation coefficient [20] to detect strong pairwise associations among features. Highly correlated features, which may represent redundant information, were considered for removal or consolidation during preprocessing to reduce noise and improve model performance. We trained a Random Forest classifier to predict the likelihood of feature request resolution using historical issue data. Random Forest was chosen because it provides a favorable balance between predictive accuracy and interpretability. The model's probability estimates allow us to assess the contribution of each feature to resolution outcomes. To interpret the model, we applied two complementary techniques: feature importance scores and Partial Dependence Plots (PDPs) [21]. Feature importance highlights the relative contribution of individual variables to model performance, while PDPs visualize the marginal effect and direction of each feature on the predicted probability of resolution. Notably, feature importance captures the strength of a feature's influence but not its direction, whereas PDPs reveal whether the effect is positive or negative. By combining these two perspectives, we obtain a comprehensive and nuanced understanding of the factors shaping feature request resolution.

TABLE I: Features selected for feature request issues.

Category	Feature	Description
Project	size	Repository size (KB)
	contributors	# of contributors for the project
	watchers	# of stars for the project
	forks	# of forks for the project
Issue	title_length	# of characters in title
	length_desc	# of characters for plain text in description
	urls	# of urls in description
	code_snippets	# of code_snippets in description
	labels	# of labels the issue has been given
Community	commments	# of comments for the issue
	creator_flwrs	# of followers of the issue creator
	asgne_contrs	# of contributions by the assignee for the issue (0 if unassigned)

Table I summarizes the features used in our analysis, grouped into three categories: project-level, issue-level, and community-level. We computed pairwise Spearman correlation coefficients to identify and remove highly correlated features, retaining a subset of relatively independent features for model construction. A Random Forest classifier was then built to predict whether a feature request was successfully resolved.

To determine the resolution status of feature requests, we used the `state` and `state_reason` fields from the GitHub REST API. Specifically, issues with `state=closed` and `state_reason=completed` were labeled as successfully resolved, while those with `state=closed` and `state_reason=not_planned` were considered explicitly unresolved. For open issues, we further distinguished between active and likely abandoned ones. Using data extracted in December 2024, we marked as unresolved any open issue created before December 2023 with no activity for at least six months. We chose the six-month window because prior work and community practice operationalize inactivity/abandonment with 3–12-month cutoffs [22]. Recent survival analyses also mark repositories as abandoned after six months without commits [23], indicating that six months is a commonly accepted cutoff in current OSS research. Issue-level dynamics reinforce this choice: studies show that most issues are resolved quickly (often days to weeks at the median), and long-lived issues are uncommon [24]. Accordingly, when an issue has been open for at least a year and then remains silent for six months, the likelihood of renewed progress is low. Treating such issues as likely abandoned is a pragmatic, literature-aligned choice that reduces outcome ambiguity in our analysis. The remaining open issues were treated as in progress and excluded from the analysis. This filtering ensured that our dataset focused on issues with reasonably definitive resolution status, thereby minimizing ambiguity in the outcome variable. As a result, we obtained 127,261 resolved and 36,835 unresolved feature requests, corresponding to a class distribution of approximately 77.5% resolved and 22.5% unresolved.

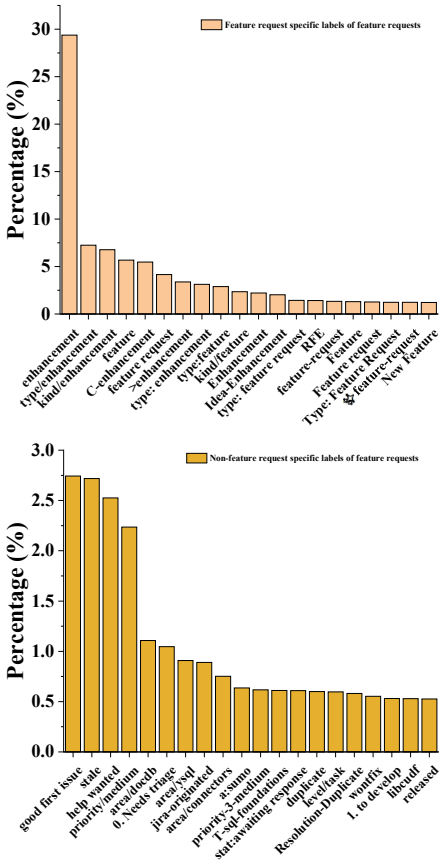


Fig. 2: Top feature request labels and co-occurring labels.

III. Results

In this section, we present the results of our study.

A. RQ1: Labeling and Evolution of Feature Requests

1) Labeling: We identified a total of 141 feature request-specific labels and 8,967 non-feature request-specific labels from the selected repositories, and recorded their frequencies. Figure 2 shows the top 10 most frequently used labels associated with feature requests. Among them, *enhancement* is the most common, accounting for 29.38% of all occurrences. This aligns with GitHub's default recommendation for labeling feature requests. The top three labels—*enhancement*, *type/enhancement*, and *kind/enhancement*—all explicitly indicate enhancements, followed closely by *feature*. These results suggest that most feature request labels are variants of *enhancement* or *feature*, often prefixed with terms such as *kind/*, *type/*, or even special symbols (e.g., stars) to indicate broader categorization schemes. In some cases, abbreviations like *RFE* (Request for Enhancement) are also used. This variation reflects diverse labeling conventions across projects, even when referring to similar types of issues.

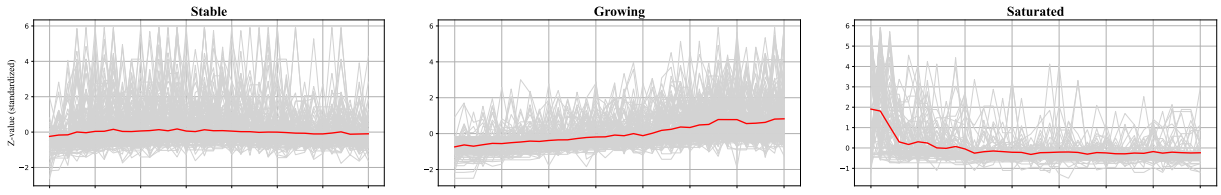
Figure 2 also shows the most frequent non-feature request labels that co-occur with feature requests. The most prevalent among them is *good first issue* (2.74%), likely due to its default availability on GitHub and widespread adoption across repositories. Its presence suggests that some feature requests are perceived as relatively simple and self-contained (e.g., UI tasks like "add a button"), making them suitable for newcomers. The *stale* label, often used for inactive issues, also appears frequently, indicating that some feature requests may face delays due to limited resources or shifting priorities. The co-occurrence of *help wanted* further indicates that some feature requests may require additional input, be underspecified, or involve greater implementation complexity, thus prompting project maintainers to seek external contributions or discussion. These patterns reflect the diverse nature of feature requests—ranging from easy-to-address items to long-standing or complex issues. Other co-occurring labels include those for issue triage and status transitions (e.g., *wontfix*), prioritization (e.g., *priority/medium*), and duplication (e.g., *duplicate*). A full list of co-occurring labels is available in the materials[1].

2) Evolution: As described in Section II-B, we collected the monthly counts of newly submitted feature requests and the cumulative counts of unresolved ones for each project. To uncover distinct temporal patterns, we applied k-means clustering to the standardized time series of these two metrics, respectively. The number of clusters was determined by evaluating clustering quality with both the Silhouette score and the balance ratio [25]. Across both tasks, the three-cluster solution provided the best trade-off between cluster separation and distributional balance. Specifically, for newly submitted feature requests, the Silhouette score was 0.46 and the balance ratio was 0.47, while for cumulative unresolved requests the scores were 0.54 and 0.67, respectively. These values indicate reasonably good compactness and separation, particularly for time series data. We also tested other solutions with two, four, and five clusters, but they yielded lower Silhouette scores and poorer balance ratios. As a result, we adopted the three-cluster solution, which produced interpretable categories of project behavior: ***Stable***, ***Growing***, and ***Saturated*** for newly submitted feature requests.

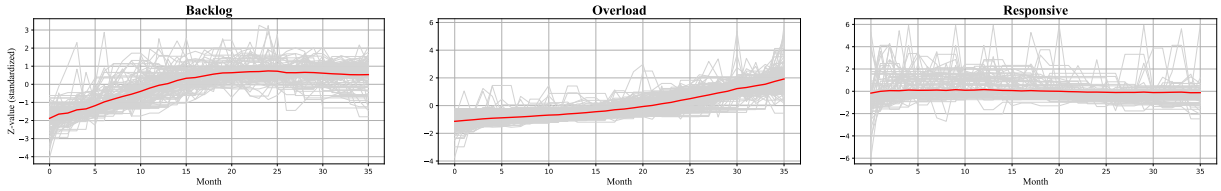
- Projects in the *Stable* category show minimal fluctuation in new feature requests, reflecting low and steady user demand. They typically have the lowest project metrics (e.g., contributor count and size), indicating small scale and limited community activity.
- The *Growing* category exhibits a steady rise in new requests, driven by increasing demand or adoption. These projects have the highest project metrics and represent fast-growing, widely adopted repositories.
- In the *Saturated* category, new requests start high but gradually decline, suggesting early growth followed by demand stabilization. These projects have moderate project metrics, reflecting a transition from expansion to maturity.

Clustering based on the monthly cumulative counts of unresolved feature requests resulted in three categories: ***Backlog***, ***Overload***, and ***Responsive***.

- In the *Backlog* category, unresolved requests grow slowly



(a) Clustering by new feature requests submissions per month.



(b) Clustering by cumulative unresolved feature requests per month.

Fig. 3: Time-series clustering of feature request behavior.

and then stabilize, indicating a stable yet persistent set of pending requests. These projects have moderately high project metrics, suggesting medium to large scale development with limited processing capacity.

- The *Overload* category shows a continuously rising backlog, reflecting increasing pressure from unresolved requests. Projects here have the highest project metrics, implying that even large and popular projects may struggle to keep pace with incoming requests due to complexity or sheer volume.
- The *Responsive* category maintains low levels of unresolved requests, reflecting timely resolution. Projects in this group typically exhibit the lowest project metrics and are often smaller in scale with manageable development workloads.

Figure 3 illustrates the temporal patterns in feature request behavior derived from clustering based on the two metrics described above. Each subfigure shows the z-score normalized time series of projects within a cluster, where thin lines denote individual projects and the bold line represents the cluster mean. Due to the large number of projects, we do not label individual project names in the figure. The proportions of projects in the *Stable*, *Growing*, and *Saturated* clusters are 42.9%, 24.2%, and 12.9%, respectively, while the *Backlog*, *Overload*, and *Responsive* patterns account for 18.6%, 34.8%, and 46.6%. These clusters reveal distinct evolution trajectories: some projects maintain a steady rhythm in feature request submission and resolution, while others constantly face a certain level of backlog or even an ever-increasing one. Such behavioral differences are closely associated with project characteristics such as size, contributor count, and popularity. Large or highly popular projects often face mounting maintenance demands and struggle to address incoming feature requests, leading to *Backlog* or *Overload* patterns. In contrast, smaller projects are more likely to sustain a balance between feature request submission and resolution, resulting in a more *Responsive* development lifecycle. The detailed feature request trends for all projects, as well as the project list for each cluster, are available in the online supplemental material[1].

Answer to RQ1

The *enhancement* label is commonly used (29.38%) to identify feature requests, but its usage varies across projects and is neither consistent nor exclusive. Feature requests frequently co-occur with labels such as *good first issue*, *stale*, and *help wanted*, indicating diverse collaboration needs. Their evolution also differs by project: larger or more popular projects tend to accumulate unresolved requests, while smaller projects are more likely to sustain balanced and responsive handling over time.

B. RQ2: Resolution and Engagement of Feature Requests

RQ2 investigates the differences between feature requests and regular issues in terms of resolution time and engagement. For resolution time, we focused on issues with a clear closure timestamp. Among all studied feature requests, 148,130 had measurable resolution times. To enable a fair comparison, we randomly sampled an equal number of closed regular issues. For engagement, we examined two metrics: the number of comments and the assignee assignment rate—that is, the proportion of issues assigned to at least one developer within a project, as introduced in Section II-C.

Figure 4 shows the distributions of feature requests and regular issues across these three metrics, using histograms (top row) and boxplots (bottom row). Feature requests generally take longer to resolve than regular issues. The median resolution time is 53 days for feature requests, compared to 8 days for regular issues. Most issues in both groups are resolved within 0–7 days, but the proportion is higher for regular issues (48.5% vs. 26.0%). To better capture the dense early resolutions and the sparse long-tail cases, we used finer time buckets in the early periods and progressively larger ones for later intervals, which improves readability and avoids

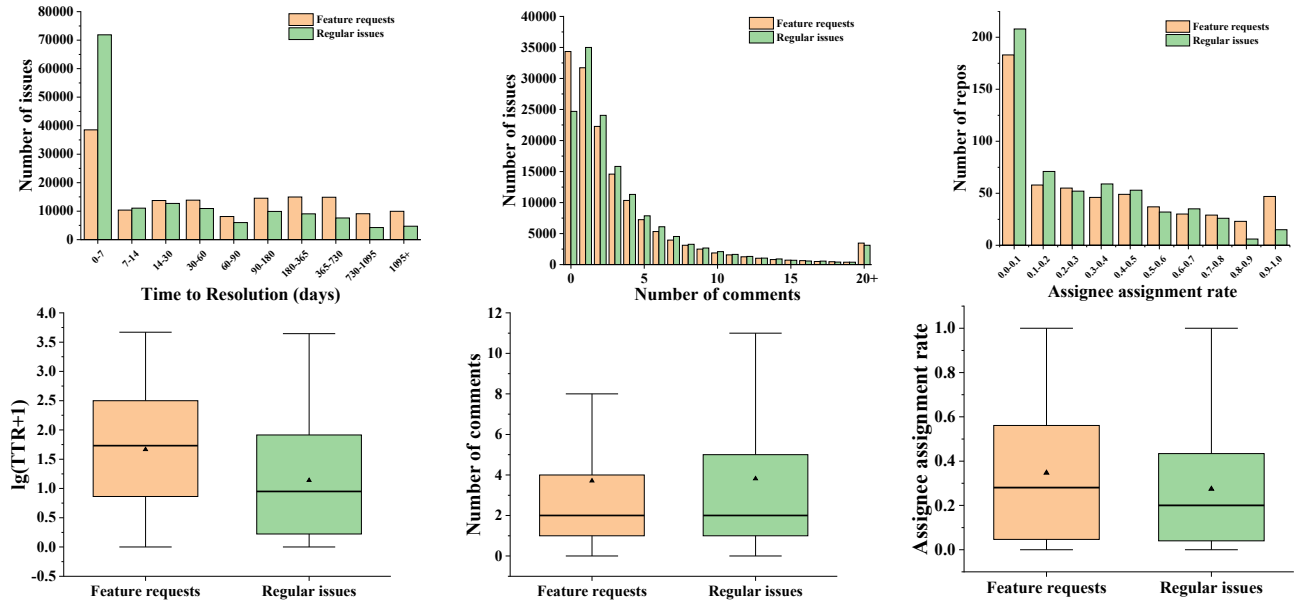


Fig. 4: Comparison of the distribution of feature requests and regular issues in resolution and engagement.

fragmented bars. Notably, 30.0% of feature requests fall into the long-resolution window (90–730 days), whereas regular issues rarely reach this range. While many feature requests are resolved quickly, a sizable fraction remains open for extended periods. This may be because they typically involve adding new functionality, which often demands more effort, coordination, and alignment with broader project goals. In contrast, regular issues, often in the form of bug reports, tend to be more urgent to resolve and therefore prioritized.

In terms of engagement, regular issues receive slightly more comments on average. Feature requests more frequently receive no comments, indicating that many do not prompt discussion. This could be because feature requests are sometimes perceived as suggestions rather than actionable tasks, and may be deferred or overlooked unless they align with current development priorities. However, feature requests are overrepresented in the 20+ comment range, indicating that some lead to extensive discussion, likely due to their complexity or strategic significance. Finally, feature requests show a higher overall assignee assignment rate than regular issues. Although most issues in both groups fall into the lowest assignment range (0–0.1), reflecting the limited availability of dedicated issue handlers in many open-source projects, feature requests are more common in the 0.8–1.0 range. This suggests that some projects actively assign feature requests to specific contributors, potentially to support roadmap-driven development or to ensure follow-through.

Additionally, we conducted statistical tests to assess whether the differences between feature requests and regular issues across the three key metrics are statistically significant. We applied the Mann–Whitney U test for *Time to Resolution* and *Number of Comments*, which involve independent samples, and the Wilcoxon Signed-Rank Test for the paired data in *Assignee Assignment Rate* (i.e., for each project, one rate for feature requests and one for regular issues) [26]. Both are non-parametric tests suitable for comparing samples without assuming normality. We report p-values along with effect sizes (r). As shown in Table II, all comparisons yielded statistically significant results. The effect sizes for *Time to Resolution* and *Assignee Assignment Rate* were small but not negligible. For *Number of Comments*, the effect size was negligible, indicating that the statistical significance may not translate into meaningful practical differences.

TABLE II: Statistical test results on differences between feature requests and regular issues.

Metric	p-value	r	Effect size
Time to resolution	$<$0.001	0.26	Small
Number of comments	$<$0.001	0.05	Negligible
Assignee assignment rate	$<$0.001	0.28	Small

Answer to RQ2

Feature requests differ from regular issues in key aspects of the resolution and engagement process. They tend to take longer to resolve and are more likely to be assigned, particularly in projects with structured issue management practices. The observed distributional differences suggest that feature requests are handled in more variable and context-dependent ways, often drawing either minimal attention or sustained coordination, warranting project-level analysis.

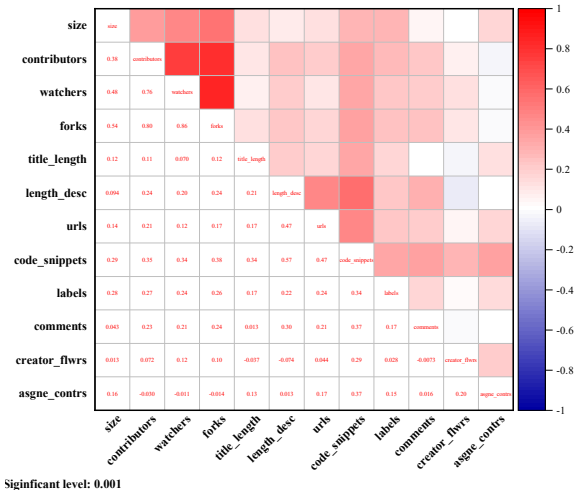


Fig. 5: The correlation analysis results among features.

C. RQ3: Factors Influencing Feature Request Resolution

This section reports the results of two analyses: a correlation analysis among the extracted features and a predictive analysis using a Random Forest model to determine whether a feature request is resolved. The correlation analysis is intended to uncover potential correlations among features, while the predictive modeling evaluates the extent to which these features contribute to the resolution of feature requests.

1) Correlation Analysis: To investigate potential intrinsic relationships among features involved in the feature request resolution process, we computed the Spearman correlation coefficients for all features introduced in Section II-D. The correlation results are visualized as a heat-map in Figure 5, where both axes list the analyzed features. The lower triangular matrix shows the pairwise Spearman coefficients, while the color gradient visually encodes the strength of these correlations—red indicates stronger correlations, whereas blue represents weaker or negligible ones. This visualization uncovers several patterns that offer insights into the structure and dynamics of feature requests.

Complex and structured feature requests often include richer contextual information: The length of the description (`length_desc`) is strongly correlated with the number of code snippets (`code_snippets`, $\rho = 0.57$) and the number of external links (`urls`, $\rho = 0.47$). In addition, the length of the title (`title_length`) exhibits a moderate correlation with `code_snippets` ($\rho = 0.34$), suggesting that more technically involved requests may exhibit more verbose titles and content.

Detailed issue descriptions tend to encourage community engagement: The number of comments (`comments`) shows moderate correlations with both `code_snippets` ($\rho = 0.37$) and `length_desc` ($\rho = 0.30$), implying that richer technical content may foster more discussion. Interestingly, the requester's social influence, measured by `creator_flwrs`, does not show a significant correlation with `comments`, suggesting that community engagement is more strongly driven by the content of the request than by the submitter's status.

Feature requests with more technical depth or submitted by influential users are more likely to attract experienced contributors: While the assignee's contribution count (`asgne_contrs`) is not correlated with repository popularity, it is positively associated with both `code_snippets` ($\rho = 0.37$) and `creator_flwrs` ($\rho = 0.20$). This indicates that technically rich requests or those submitted by prominent users tend to receive attention from highly active contributors.

Based on the correlation analysis, we also identified and removed highly correlated features that may introduce redundancy in the training process of the Random Forest classifier. Specifically, features with a Spearman correlation coefficient greater than 0.7 were excluded to reduce redundancy and improve model performance. Among all features, only `watchers` and `forks` met this criterion due to their strong correlation with `contributors`, and thus were removed, leaving `contributors` retained. This step also helps ensure the reliability of the subsequent PDPs, which analyze the influence of individual features.

2) Predictive Analysis: We trained a Random Forest classifier to predict whether a feature request would be successfully resolved. The dataset was split into training and testing sets using an 80/20 ratio with stratification to preserve the original class distribution. The classifier was configured with the following hyperparameters: `n_estimators=400`, `max_depth=None`, `min_samples_split=2`, `min_samples_leaf=1`, `max_features='sqrt'`, `bootstrap=True`, `criterion='gini'`, and `random_state=42`. We experimented with different hyperparameter settings and selected this configuration based on consistent empirical performance across validation runs. On the held-out test set, evaluated with stratified 10-fold cross-validation, the model achieved an accuracy of 0.8212, precision of 0.8443, recall of 0.9433, F1-score of 0.8911, and AUC of 0.8355, demonstrating strong predictive performance for this task. Based on this model, we analyzed the relative influence of features on resolution outcomes using two complementary interpretability techniques: feature importance scores and PDPs. Figure 6 presents the feature importance rankings, and Figure 7 presents PDPs for a subset of features whose marginal effects exhibit clear and representative patterns. These features were selected to best illustrate the model's behavior. PDPs for additional features are provided in the supplementary materials[1].

As shown in Figure 6, the length of the description and title (indicators of request complexity) are among the most important features. However, the PDPs in Figure 7 show that resolution likelihood rises with description and title length up to a moderate point, then falls as verbosity grows further, reflecting an overall negative effect at higher complexity. This suggests that although a certain level of detail is beneficial, excessive complexity may hinder successful resolution. Similar patterns are observed for content elements such as URLs and code snippets. Although they rank lower in importance, their marginal effects are also negative. This does not imply

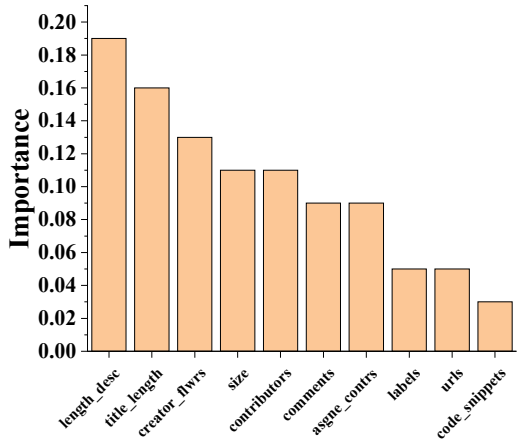


Fig. 6: Importance scores for features.

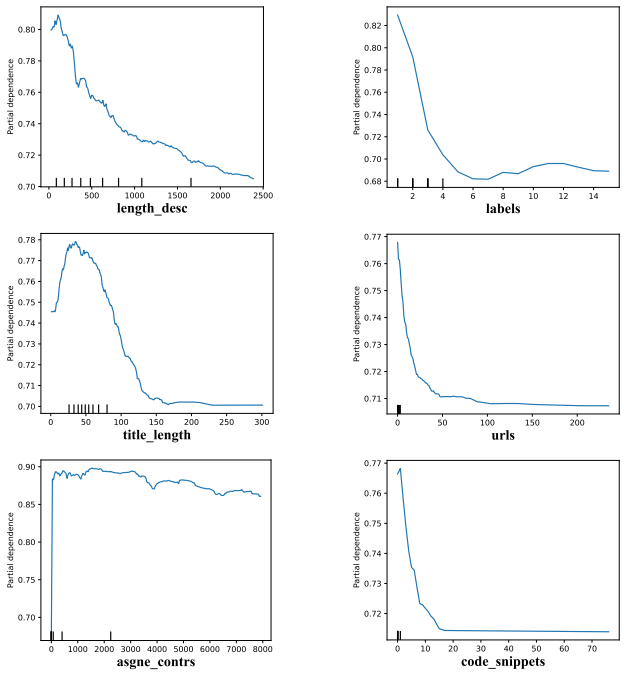


Fig. 7: PDPs for selected features with clear marginal effects.

these elements hinder resolution outright, but rather that the additional detail they provide may not always compensate for the increased complexity or potential ambiguity. This finding aligns with the intuition that overly lengthy or complex requests may be harder to understand or address, either due to vague articulation or because they span multiple concerns.

In contrast, features related to the reporter's experience and reputation, specifically the number of followers and contributions, rank among the relatively more influential predictors and exhibit a positive effect on resolution likelihood. Compared to features capturing community interaction or assignee experience (e.g., number of comments and assignee contributions), the experience and reputation of the reporter appear to have a stronger influence on whether a request is addressed. Nonetheless, the number of comments also shows a positive association, suggesting that community engagement contributes to successful outcomes. Although a high comment count may signal complexity, the benefits of discussion appear to outweigh this drawback. Likewise, the assignee's prior contributions positively influence resolution. The PDP analysis indicates a noticeable increase in resolution likelihood when the assignee has any prior involvement with the repository, highlighting the value of contributor familiarity with the project.

Features capturing static project characteristics, such as project size and number of contributors, fall between the two aforementioned groups in importance. These features are positively associated with resolution outcomes, suggesting that more active projects are better positioned to address feature requests. In summary, request complexity and reporter credibility emerge as the most influential factors, followed by project context, community engagement, and assignee experience. These findings highlight the importance of clear communication and suggest that fostering inclusive contributor dynamics may improve feature request resolution.

Answer to RQ3

Our analysis indicates that both request complexity and contributor profiles shape feature-request outcomes. Requests with longer titles and descriptions—often containing code snippets and external links—are less likely to be addressed, reflecting the cognitive load of overly detailed reports. By contrast, requests submitted by reporters with a strong contribution history and follower base, assignments to active contributors, and lively community discussions boost the likelihood of resolution. Project-level factors, such as a higher contributor count, further support resolution.

IV. Implications

Project maintainers. Our findings reveal that the labeling of feature requests varies considerably across projects and often lacks consistency. To improve clarity and communication, maintainers should adopt coherent labeling strategies. At the same time, not all feature requests warrant immediate resolution—some may represent vague or low-priority ideas that are unlikely to be addressed in the near term. What is important, however, is transparent communication of prioritization decisions. Maintainers can, for example, explicitly label low-priority or non-actionable requests and indicate whether requests are accepted, deferred, or declined, along with brief explanations (e.g., resource constraints or misalignment with project goals). Such transparency helps manage contributor expectations and reduce uncertainty. In addition, projects may benefit from periodic backlog hygiene, including reviewing and removing outdated or inactive requests, merging duplicates, and re-evaluating priorities, es-

pecially in large repositories where long-term accumulation can hinder productivity. Finally, since resolution outcomes are also influenced by developer-level factors such as prior contributions and assignment, maintainers can assign higher-priority requests to developers with relevant expertise and a history of contributions in the affected components, thereby improving resolution efficiency.

Request submitters. Our findings suggest that lengthy or complex feature requests are less likely to receive timely responses. Submitters should focus on clear and specific descriptions. When possible, they should break down complex requests into smaller, more manageable parts. For large or active projects, where request backlogs are common, submitters are advised to check for similar existing requests before submitting a new one and provide concrete examples or motivations to help maintainers understand the request. Submitters should also engage in follow-up discussions and respond to maintainer questions. Active participation increases the chance of the feature request being considered and implemented.

Researchers. The inconsistent labeling of feature requests presents opportunities for researchers to develop tools that automatically identify and categorize them using natural language processing techniques. Tools that help submitters refine and simplify their requests could also improve resolution outcomes. For large-scale projects, there is a need for methods to support the prioritization and assignment of feature requests. Future work can explore how to automatically match requests with suitable developers based on past activity or expertise. Our results and dataset also offer a basis for studying broader aspects of request management, such as triage practices and contributor coordination.

V. Threats to Validity

Limited repository selection. Our dataset was constructed from public GitHub repositories written in the ten most popular programming languages on GitHub, requiring at least 20 stars and one commit by December 2024. While these criteria may bias the dataset toward projects with some visibility and maintenance activity, they also filter out personal or abandoned repositories that provide little analytical value. From this pool, we randomly sampled 100 repositories per language, resulting in a diverse mix of activity levels. Our analysis showed that the core evolution and resolution patterns of feature requests were consistent across projects with varying activity levels, which mitigates concerns about generalizability. Nonetheless, we acknowledge that our findings may be less representative of very small, private, or niche projects. To further support generalization, we release our dataset and scripts so that future studies can replicate our analysis on different types of repositories.

Threats in feature request identification. Our method for identifying feature requests combines label-based extraction (e.g., "enhancement" and "feature request") with keyword filtering on issue titles and bodies. Given inconsistent labeling practices, some feature requests could be overlooked or misclassified. We addressed this by manually reviewing all labels to build a comprehensive list, applying a secondary keyword search on unlabeled issues, and then manually validating those candidates. This extra step yielded only 1,499 additional cases, indicating that any remaining misclassification is unlikely to alter our main patterns.

Feature selection bias. To study what drives resolution, we picked features at the project, issue, and community levels based on prior research. Yet, other factors might also matter. We performed a Spearman correlation analysis to remove highly redundant variables and ensure coverage of different dimensions of project dynamics, issue characteristics, and community behavior. In addition, we briefly tested including code-related metrics (e.g., lines changed per pull request) and newcomer activity counts; neither improved predictive performance or interpretability, so we retained our original feature set. Future work can explore these and other variables in more depth.

Predictive-model limitations. In RQ3, we selected Random Forest for predictive analysis to better understand the factors influencing feature request resolution. While other models could have been used, we also evaluated several alternatives, including logistic regression and decision trees, but found that Random Forest (F1 = 0.89) consistently achieved the best performance. For interpretability, we combined feature importance scores with PDPs: importance scores identify which predictors most strongly affect model performance, while PDPs illustrate how each feature influences the likelihood of resolution. This combination provides a more comprehensive and transparent view of how different factors shape feature request outcomes.

VI. Related Works

We review related work in two areas: studies on specific types of GitHub issues and research on analyzing or improving issue-tracking practices.

A. Studies on Specific GitHub Issue Types

GitHub provides a set of default issue labels, and prior studies have analyzed specific types of labeled issues to understand their characteristics, management practices, and impact on software development. Among these, *good first issues* (GFIs) have received significant attention for their role in onboarding newcomers. Huang et al. [16] proposed a machine learning model to predict GFIs based on 79 features related to the semantics, readability, and text richness. Tan et al. [10] analyzed over 9,000 GFIs and found that nearly half are not solved by newcomers, often due to vague descriptions or skill mismatches. They also identified best practices, such as clear documentation and technical accessibility. Building on this line of work, Xiao et al. [27] developed RecGFI, a system that recommends suitable GFIs to newcomers. Horiguchi et al. [28] further examined the current state of the GFI mechanism and its impact on newcomer retention. *Bugs* have also been widely studied, with a focus on improving triage and resolution. Huang et al. [29] incorporated social and technical features, such as contributor sentiment and experience, to predict bug

report priority. Ma et al. [30] explored cross-project bugs in the scientific Python ecosystem, highlighting practices for root cause analysis and coordinated resolution. Sohrawardi et al. [31] compared five different algorithms to classify the issue reports into bug and non-bug. *Wontfix* issues, which are closed without resolution, have drawn increasing attention. Panichella et al. [32] investigated the causes of wontfix issues and the factors influencing their resolution time, compared three machine learning methods to predict whether an issue would be labeled as wontfix. Similarly, Han et al. [33] and Wang et al. [34] extracted and analyzed the characteristics of wontfix issues and built models to predict them.

In contrast, research on *feature requests* remains relatively limited. Hu et al. [35] compared bugs and feature requests in terms of discussion dynamics and resolution delays, while Nyamawe et al. [36] developed a machine learning approach that recommends refactorings for newly submitted feature requests based on the history of the previously requested features, applied refactorings, and code smells information. Outside GitHub, Heppler et al. [37] examined feature requests in the Eclipse IDE, analyzing implementation likelihood based on factors such as reporter identity and priority. Despite their importance as a communication channel between users and maintainers, feature requests remain understudied. Our study addresses this gap through a large-scale empirical study of feature requests in GitHub repositories.

B. Analyses and Enhancements of GitHub Issue Tracking

Beyond studies targeting issues with specific labels, recent work has taken a broader perspective on GitHub's issue-tracking ecosystem, exploring how labeling practices, issue templates, and prioritization strategies affect developer workflows and resolution efficiency. While labeling plays a key role in organizing issues, its application varies significantly across projects. For instance, Bissyandé et al. [6] surveyed a large number of repositories and found that bug and feature request are the most frequently used labels. However, GitHub's flexible, project-specific labeling system often results in inconsistent usage. Nascimento et al. [38] analyzed both default and custom labels, showing that teams frequently refine their label sets to reflect changing priorities, with the timing of label updates often indicating shifts in focus. Kim et al. [11] showed that ad hoc multi-label schemes can introduce ambiguity and lead to misclassification, highlighting the lack of attention to multi-label classification in current automated labeling tools such as TicketTagger [39].

Templates and prioritization mechanisms also influence how issues are managed. Sülün et al. [9] analyzed over one million issues and found that nearly all large-scale projects adopt templates for bug reports and feature requests. They observed that issues created using templates are less likely to be reopened, indicating clearer initial descriptions. Li et al. [40] conducted a large-scale mixed-methods study on issue and pull-request templates. They found that templates often welcome contributors, structure the submitted content, and, after adoption, lead to reductions in both the number of incoming issues or pull requests and the volume of discussion comments. In terms of prioritization, Noei et al. [41] leveraged user reviews from mobile app markets to rank open-source Android app issues on GitHub. He et al. [12] investigated how priority labels are currently used in practice and found that existing labeling mechanisms provide limited support for indicating issue urgency. To improve issue management, researchers have applied machine learning and natural language processing techniques to automate tasks such as label prediction and priority estimation. For example, Fan et al. [42] proposed an automated classification approach based on a two-stage framework. Izadi et al. [43] developed a method combining feature engineering with textual classifiers to predict issue purpose and priority, streamlining issue triage. Siddiq et al. [44] showed that a BERT-based model can effectively distinguish between bugs, enhancements, and questions.

Despite these advances, feature requests remain underexplored. Although Sülün et al. [9] noted that templates for feature requests are less standardized than those for bugs, the impact of this inconsistency on resolution and implementation outcomes is still unclear. Moreover, most automation efforts treat feature requests as a generic issue type, overlooking their distinct lifecycle—from initial proposal and community discussion to eventual resolution. To fill this gap, our study offers the first large-scale, issue-type-specific empirical analysis of feature requests on GitHub, examining how they are labeled, how they evolve, and how they are ultimately addressed.

VII. Conclusion

This study presents a large-scale empirical analysis of feature request issues on GitHub, examining their labeling practices, resolution characteristics, and influencing factors. We find that labeling is inconsistent across projects, feature requests are generally resolved more slowly than other issues, and a growing backlog is common in large or popular repositories. Request complexity and contributor profiles emerge as key factors shaping acceptance outcomes, with overly lengthy descriptions linked to lower resolution rates. These results highlight the importance of clear communication for request submitters and proactive triage by maintainers.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 62372232, 62372227).

References

[1] GitHub, "Github docs." https://docs.github.com, 2025.
[2] A. Ajibode, D. Yunwei, and Y. Hongji, "Software issues report for bug fixing process: An empirical study of machine-learning libraries," 2023.
[3] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 392–401, 2013.
[4] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Predicting issue types on github," *Science of Computer Programming*, vol. 205, p. 102598, 2021.
[5] H. Borges, R. Brito, and M. T. Valente, "Beyond textual issues: Understanding the usage and impact of github reactions," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES '19, (New York, NY, USA), p. 397–406, Association for Computing Machinery, 2019.

[6] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "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)*, pp. 188–197, IEEE, 2013.

[7] T. Chen, Y. Zhang, S. Chen, T. Wang, and Y. Wu, "Let's supercharge the workflows: An empirical study of github actions," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 01–10, IEEE, 2021.

[8] J. Liu, J. Li, and L. He, "A comparative study of the effects of pull request on github projects," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 313–322, 2016.

[9] E. Sülün, M. Saçakçı, and E. Tüzün, "An empirical analysis of issue templates usage in large-scale projects on github," *ACM Transactions on Software Engineering and Methodology*, 2024.

[10] X. Tan, M. Zhou, and Z. Sun, "A first look at good first issues on github," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, (New York, NY, USA), p. 398–409, Association for Computing Machinery, 2020.

[11] J. Kim and S. Lee, "An empirical study on using multi-labels for issues in github," *IEEE Access*, vol. 9, pp. 134984–134997, 2021.

[12] Y. He, W. Yang, M. Pan, Y. Hussain, and Y. Zhou, "Understanding and enhancing issue prioritization in github," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 813–824, 2023.

[13] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*, pp. 291–302, 2016.

[14] R. Kikas, M. Dumas, and D. Pfahl, "Issue dynamics in github projects," in *Product-Focused Software Process Improvement: 16th International Conference, PROFES 2015, Bolzano, Italy, December 2-4, 2015, Proceedings 16*, pp. 295–310, Springer, 2015.

[15] 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.

[16] Y. Huang, J. Wang, S. Wang, Z. Liu, D. Wang, and Q. Wang, "Characterizing and predicting good first issues," in *Proceedings of the 15th ACM/IEEE international symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–12, 2021.

[17] J. Han, S. Deng, X. Xia, D. Wang, and J. Yin, "Characterization and prediction of popular projects on github," in *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*, vol. 1, pp. 21–26, IEEE, 2019.

[18] W. Xiao, J. Li, H. He, R. Qiu, and M. Zhou, "Personalized first issue recommender for newcomers in open source projects," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 800–812, IEEE, 2023.

[19] V. K. Eluri, T. A. Mazzuchi, and S. Sarkani, "Predicting long-time contributors for github projects using machine learning," *Information and Software Technology*, vol. 138, p. 106616, 2021.

[20] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, pp. 2146–2189, 2016.

[21] R. Couronné, P. Probst, and A.-L. Boulesteix, "Random forest versus logistic regression: a large-scale benchmark experiment," *BMC bioinformatics*, vol. 19, pp. 1–14, 2018.

[22] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, "On the abandonment and survival of open source projects: An empirical investigation," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–12, 2019.

[23] A. M. Thakur, R. Milewicz, M. Jahanshahi, L. Paganini, B. Vasilescu, and A. Mockus, "Scientific open-source software is less likely to become abandoned than one might think! lessons from curating a catalog of maintained scientific software," *Proc. ACM Softw. Eng.*, vol. 2, June 2025.

[24] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, (New York, NY, USA), p. 291–302, Association for Computing Machinery, 2016.

[25] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 747–748, 2020.

[26] C. O. Fritz, P. E. Morris, and J. J. Richler, "Effect size estimates: current use, calculations, and interpretation.," *Journal of experimental psychology: General*, vol. 141, no. 1, p. 2, 2012.

[27] W. Xiao, H. He, W. Xu, X. Tan, J. Dong, and M. Zhou, "Recommending good first issues in github oss projects," in *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, (New York, NY, USA), p. 1830–1842, Association for Computing Machinery, 2022.

[28] H. Horiguchi, I. Omori, and M. Ohira, "Onboarding to open source projects with good first issues: A preliminary analysis," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 501–505, 2021.

[29] Z. Huang, Z. Shao, G. Fan, H. Yu, K. Yang, and Z. Zhou, "Bug report priority prediction using social and technical features," *Journal of Software: Evolution and Process*, vol. 36, no. 6, p. e2616, 2024.

[30] W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu, "How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 381–392, IEEE, 2017.

[31] S. J. Sohrawardi, I. Azam, and S. Hosain, "A comparative study of text classification algorithms on user submitted bug reports," in *Ninth International Conference on Digital Information Management (ICDIM 2014)*, pp. 242–247, IEEE, 2014.

[32] S. Panichella, G. Canfora, and A. Di Sorbo, ""won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on github," *Information and Software Technology*, vol. 139, p. 106665, 2021.

[33] Z. Han, B. Jiang, W. Xue, C. Dai, Q. Huang, and Y. Wang, "A multi-agent collaboration approach for identifying developer-fixed issues in github projects," in *CCF Conference on Computer Supported Cooperative Work and Social Computing*, pp. 459–470, Springer, 2024.

[34] Y. Wang, Z. Han, Q. Huang, and B. Jiang, "Why not fix this bug? characterizing and identifying bug-tagged issues that are truly fixed by developers," *Journal of Software: Evolution and Process*, vol. 37, no. 2, p. e70008, 2025.

[35] D. Hu, T. Wang, J. Chang, Y. Zhang, and G. Yin, "Bugs and features, do developers treat them differently?," in *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pp. 250–255, IEEE, 2018.

[36] A. S. Nyamawe, H. Liu, N. Niu, Q. Umer, and Z. Niu, "Feature requests-based recommendation of software refactorings," *Empirical Software Engineering*, vol. 25, pp. 4315–4347, 2020.

[37] L. Heppler, R. Eckert, and M. Stuermer, "Who cares about my feature request?," in *Open Source Systems: Integrating Communities: 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30-June 2, 2016, Proceedings 12*, pp. 85–96, Springer, 2016.

[38] L. P. Nascimento, A. Santos, I. Machado, *et al.*, "Issue labeling dynamics in open-source projects: A comprehensive analysis," in *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*, pp. 51–60, SBC, 2024.

[39] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 406–409, 2019.

[40] Z. Li, Y. Yu, T. Wang, Y. Lei, Y. Wang, and H. Wang, "To follow or not to follow: Understanding ¡italic¿issue/pull-request templates¡/italic¿ on github," *IEEE Trans. Softw. Eng.*, vol. 49, p. 2530–2544, Apr. 2023.

[41] E. Noei, F. Zhang, S. Wang, and Y. Zou, "Towards prioritizing user-related issue reports of mobile applications," *Empirical Software Engineering*, vol. 24, pp. 1964–1996, 2019.

[42] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 121–130, IEEE, 2017.

[43] M. Izadi, K. Akbari, and A. Heydarnoori, "Predicting the objective and priority of issue reports in software repositories," *Empirical Software Engineering*, vol. 27, no. 2, p. 50, 2022.

[44] M. L. Siddiq and J. C. Santos, "Bert-based github issue report classification," in *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, pp. 33–36, 2022.