On the Impact of Draft Pull Requests on Accelerating Feedback

Firas Harbaoui
ETS - Quebec University
Montreal,QC
firas.harbaoui.1@ens.etsmtl.ca

Mohammed Sayagh
ETS - Quebec University
Montreal,QC
mohammed.sayagh@etsmtl.ca

Rabe Abdalkareem
Omar AL-mukhtar University
Al Bayda, Libya
rabe.abdalkareem@omu.edu.ly

***Abstract*—The pull request (PR) mechanism provides a structured way for developers to present their proposed modifications, engage in code review, and address any concerns before the changes are incorporated into the main codebase. As the adoption of PRs has grown over time, a variant known as draft PRs has gained traction, offering developers a novel way to share ideas and get feedback and collaboration on code changes during their development. Despite the benefits that draft PRs offer, there is still a lack of research on their efficiency in reaching the goal for which they are conceived, which is getting early feedback. This research paper aims to fill this gap by exploring how the draft mechanism is used, the impact of the draft mechanism on getting feedback and on the integration of new changes, and the different factors contributing to the responsiveness of comments. We observe that the draft is used by practitioners for complex changes and for different purposes, including the discussion of features to implement, experiments to conduct, and new versions to release. However, the goal of the draft mechanism is not fully reached as practitioners do not receive feedback (i.e., issue or review comments) on their drafts. For that, we leverage explanatory machine learning models to understand the differences between drafts that receive comments from these that did not before becoming ready to review. Our models show a median AUC performance of 0.67 to 0.77 and 0.66 to 0.75 for the issue comments and review comments models, respectively. The interpretation of our models shows that the actions that developers take as events, the description of the draft, the engagement of the author of the draft, and the engagement and responsiveness of the main reviewers play a crucial role in encouraging the reception of comments for draft pull requests.**

***Index Terms*—pull requests, Draft pull requests, Empirical Software Engineering**

I. Introduction

The Github pull request (PR) is a mechanism for developers to get feedback on their changes before merging them. For instance, developers submit their code changes and receive feedback from their peers. Such feedback can take different forms, such as suggestions to improve the code, discussions around the change, and even simple reactions. According to the obtained comments, one has to improve the changes through different revisions before the change gets merged into the main branch and eventually get released in the new version of a system.

To improve the feedback, GitHub introduced a mechanism that allows practitioners to obtain early feedback on an idea before even implementing it or on an ongoing change. Such a mechanism is called **Draft PRs**. This allows developers to get early feedback when they are still working on the change, attract collaborations, discuss ideas before starting their implementations, run different experiments, and discuss them with other developers. For instance, Github reported the following:

"At GitHub, we've always felt that you should be able to open a PR to start a conversation with your collaborators as soon as your brilliant idea or code is ready to take shape ... what if you want to signal that a PR is just the start of the conversation and your code isn't in any state to be judged? ... you'd still like people to check it out locally and give you feedback. Or perhaps you've opened a PR without any code at all in order to get the discussion started" [1].

While a large body of work has investigated the PR mechanism, no prior studies focused on the draft mechanism. Prior studies focused on different angles of the PRs, including the characteristics of abandoned PRs [2], the PR review process [3] and its impact on open source software projects [4]. However, no prior work studied the draft mechanism to better understand whether it reaches its objective (getting feedback), and how to improve such a mechanism so practitioners get better feedback on their drafts.

Thus, the goal of this paper is to fill such a gap by studying the draft mechanism . In this paper, we conduct an empirical study on seven popular software systems to better understand how the draft mechanism is used in practice, whether the goal for which the draft mechanism is conceived is reached, and how can further improve the benefit of the draft through explainatory machine learning. In particular, we address the following research questions:

- **RQ1. How do developers use the draft mechanism?** We observe that practitioners use the draft mechanism with source code changes so they rarely use the drafts just for discussing ideas (as was the goal of Github behind the draft mechanism). We also observe that drafts are associated with larger (3.72 to 7.82 times) changes compared to ordinary PRs. Through a qualitative study, we observe that the drafts are used for different purposes that can be summarized into three themes: improving features, experimenting with new ideas, and carrying out release-related activities.

- **RQ2. What is the impact of drafts on feedback and integration of changes?** We observe that the draft mechanism requires further attention for reaching the goal for which they were conceived. In fact, only 12% to 39.9% of the drafts received comments during the draft period. Merging draft PRs (from the time a draft is ready for review to its merge) requires between 18.6 and 86.6 hours, while it is only 3.2 and 24.8 hours for ordinary PRs.

Since we observe that the draft mechanism is commonly used for complex changes, while drafts often do not receive comments, we leverage an explanatory machine learning model to better understand what distinguish between drafts that receive comments from the drafts that do not. In particular, we address the following research questions:

- **RQ3. How performant are our classification models in classifying draft PRs?** We observe that our explanatory models reach good performances for classifying drafts accroding to their reception of issue comments (a median AUC between 0.67 and 0.77) and review comments (a median AUC between 0.66 and 0.75), whereas all of our studied dimensions show enough explanatory power. We observe that the most important dimension is related to the actions developers take, as a sort of events, whereas in most of the cases the project dimension, which represents the context, is the less important dimension.
- **RQ4. What are the most important features that distinguish drafts that receive comments from drafts that do not receive comments?** We observe from the interpretation of our models that one has to explicitly mention other developers or ask for comments. Drafts that are complex and have a detailed description are more likely to receive a comment. Being involved in other pull requests as well as the availability of the most important integrators are also two important factors that increase the chances to receive a comment.

We provide a replication package [1] that includes data and used scripts in our study.

This paper is structured as follows. Section II discusses our methodology. Section III discusses our empirical results on the usage of the draft mechanism. Section IV discusses the results of our explanatory machine learning models. Section V discusses the implication of our results. Section VI discusses the closest work to our paper. Section VII discusses our threats to validity and Section VIII concludes the paper.

II. Methodology

This section describes the methodology we followed to answer our research questions. In particular, we discuss the selection of the projects we studied (Section II-A), how we identify draft PRs and PRs that were drafts (Section II-B), and how we collect data to answer our four research questions (Section II-C). Note that in each research question, we further describe its own approach.

[1]https://zenodo.org/records/12796675

A. Projects Selection

Project Name	# Opened PRs	# Closed PRs	# Opened Drafts	# Closed Drafts	# Developers	# Stars
Nixos/nixpkgs	3,258	146,126	685	1,307	4,337	9.7k
Sourcegraph/Sourcegraph	348	22,381	125	753	270	6.1k
dotnet/runtime	274	20,478	44	756	1,921	9.1k
cockroachdb/cockroach	796	39,951	221	550	574	24.7k
WordPress/gutenberg	859	21,068	232	613	915	6.9k
elastic/kibana	642	87,312	429	2,215	753	17.5k
ampproject/amphtml	417	23,843	160	561	1,094	14.9k

TABLE I: List of projects used in our study

In this study, we focus on seven projects that are shown in Table I. To obtain these projects, we first collect the 15,000 top-rated projects from GitHub to avoid non-popular or toy projects. We then sort these projects based on the number of drafts they have. Those are only the PRs that are still drafts at the collection time of these projects or drafts that were closed as drafts but did not become ordinary PRs since we cannot identify from the GitHub API which PRs were drafts before at a large scale of thousands of projects. We select the top-10 projects. Since we wish to also study the drafts PRs that were merged and the impact of the draft mechanism on merging PRs (part of RQ1), we excluded three projects as we cannot distinguish their merged from rejected PRs. They all end up being with the status closed. We end up with seven projects that are popular, actively maintained and use enough drafts that we can study. As shown in Table I, our studied projects have at least 6.1k stars and at least 270 contributors.

B. Identification of Drafts Pull-Requests

While the GitHub API provides which PRs are still drafts or closed as drafts, the API does not provide information about which PRs were drafts before becoming an ordinary PR. To identify which PRs were drafts, we crawl the HTML pages of the PRs and search for the event "marked this pull request as ready for review", which marks the time at which the draft PR switched to the ordinary PRs. Based on that event, we split the PR into two phases, the first one is when the PR was a draft and the second phase is when that draft becomes an ordinary PR. Since a draft pull request can be originally created as ordinary pull request, then converted to drafts, we excluded these cases to focus only on draft pull requests that were created with the intention of being drafts. To do so, we exclude all drafts with the events "convert_to_draft" before the "ready_for_review" event as well as drafts that are still drafts (i.e., either started as drafts and remained drafts or started as ordinary pull request and converted to drafts). We end up by studying a total number of 20,564 draft pull requests distributed as follows: 1,199, 1,293, 1,500, 9,581, 1,270, 3,639, 2,082 drafts in ampproject/amphtml, cockroachdb/cockroach, dotnet/runtime, elastic/kibana, NixOS/nixpkgs, sourcegraph/-sourcegraph, and WordPress/gutenberg.

C. Data Selection

To answer our research questions, we collected different data from the Github API and by crawling the HTML pages of the draft PRs, as shown in Figure 1. We mainly use the

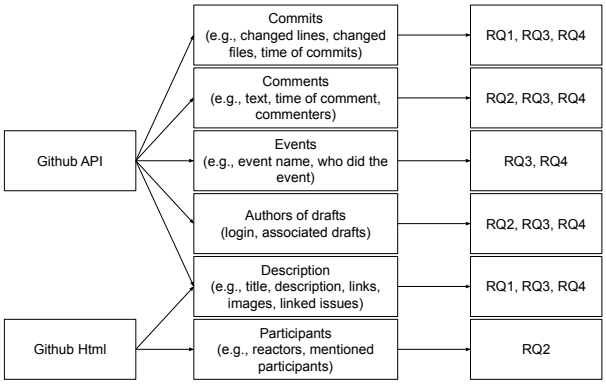


Fig. 1: Used dataset to answer our research questions.

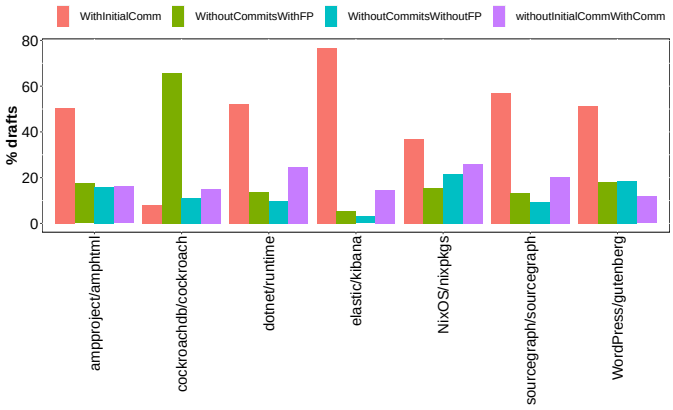


Fig. 2: Percentages of drafts with initial commits (i.e. Red bars), drafts without initial commits but commits during the draft period (i.e. Violet bars), drafts without any commits at all but with force pushes (i.e. Green bars), and drafts without any commits at all and without force pushes (i.e. Blue bars)

Github API to collect our data, except for data that are not provided through the API, such as the time a draft became ready for review. Note that we further discuss our approaches in the approach section of each RQ.

III. Understanding the Usage of the Draft PRs

The objective of our study is to empirically understand how the draft mechanism is used in practice. In particular, we wish to understand whether the draft is commonly used, how the draft mechanism is used, and the impact of the drafts on receiving feedback and integrating new changes. To do so, we address the following research questions:

- **RQ1. How do developers use the draft mechanism?**
- **RQ2. What is the impact of drafts on feedback and integration of changes?**

RQ1. How do developers use the draft mechanism?

Motivation: The goal of this research question is to quantify the popularity of the draft mechanism and qualitatively and quantiatively study for what types of changes do developers use the drafts mechanism to better understand whether it is worth exploring and improving such a mechanism or whether it is abandoned by developers, identify whether the draft is used for what is conceived for (getting feedback on ideas or source code modifications), and for what purposes do practitioners use the draft mechanism. Our results will motivate future work further investigate the draft mechanism and Github better understand how the draft mechanism is used in practice.

Approach: In this research question, we conduct the three following analyses:

(1) How practitioners use the draft mechanism: Since a draft can be created for two purposes, i.e., getting feedback on a code change or getting feedback on ideas, we quantify whether practitioners open drafts with initial commits to get feedback on them, open a draft with no commits but add them after a discussion, or they open and close drafts with no source code modifications to discuss ideas. We also identify whether drafts are used for more complex code changes than ordinary PRs by looking at the size of changes made in the draft mechanism and ordinary PRs.

(2) Types of changes associated with the draft mechanism: We leverage quantitative and qualitative analysis to understand for what types of changes do developers use the draft mechanism. To do so, we leverage the Latent Dirichlet Allocation (LDA) algorithm to identify 16 topics, which we identify using the following steps:

- Data Preprocessing: We preprocess the draft descriptions by removing stop words and punctuations, and performing stemming and lemmatization.

- Topic Modeling and Assigning: We used the Mallet [5] implementation to generate a set of topics and their corresponding keywords. To determine the number of topics, we manually tested various number of topics to determine the most relevant and coherent ones, similar to prior studies [6], [7]. In particular, we increase the number of topics when the generated topics are less cohesive and reduce the number of topics when two or more topics are similar. After careful analysis, we selected 16 topics that best represent the prevalent themes found in the draft PR descriptions.

- Quantifying the number of drafts for a given topic: To identify the popularity of different topics, we count the number of drafts for each topic. A draft X is related to a given topic Y when that draft membership probability to topic Y is greater than 10% (that is, a threshold defined by Barua et al. [8]).

- Manual Analysis: To better understand what practitioners discuss around our 16 identified topics and label them, we manually analyze the top 5 to 10 drafts of each topic. We end up manually studying 104 different drafts.

Results: Draft PRs are commonly used in our evaluated projects mainly for sharing source code modifications. We observe four types of usage: (1) drafts opened with initial commits, (2) drafts without initial commits but received commits before they were ready for review, (3) drafts that did not receive any commits at all but receive force push modifications after that and (4) drafts that did not receive any commits nor force pushes. We found that the most common category for six out of seven studied projects is drafts with

initial commits (39.6% to 76.8%), as shown in Figure 2. Additionally, we observe that 23.2% to 91.9% of the drafts that were created do not have any initial source code commits. On the other side, 3.03% to 18.5% of the drafts did not receive any commits nor force push changes that can hints into the fact that these drafts are to discuss ideas instead of source code modifications.

We also observe that 24.3% to 62.9% of drafts created without initial commits were updated with at least one commit during the draft period before becoming ready for review. These updates occurred after a median ranging from 11.7 to 44.1 hours. Interestingly, we also found that a considerable portion of the comments - between 15.96% to 61.59% - received before these drafts are ready for review actually occur before their first commit. This suggests that developers are actively seeking early feedback on their ideas after creating the draft and before coding.

Drafts are used for more extensive code changes and development compared to ordinary PRs, suggesting more collaboration and feedback from reviewers when using the draft mechanism. In fact, drafts contain a median of 3.72 to 7.82 times more changed lines than ordinary PRs, as shown in Figure 3. The median of the total number of changed lines in drafts ranges between 33 to 359, while it is between 6 to 66 for ordinary PRs. Note that such a difference is statistically significant (Wilcoxon test; p-value < 0.01). In addition to that, our analysis uncovers interesting insights regarding the length and nature of draft PRs compared to ordinary ones. We find that drafts tend to have significantly lengthier descriptions, containing a median of 1.34 to 3.25 times more words than ordinary PRs in six different projects. This difference is significant for six out of the seven projects, as confirmed by a Wilcoxon test ($p - value < 0.01$).

Draft PRs serve diverse purposes and are employed for various tasks related to enhancement of existing features, experimenting new ideas, and release-related activities. Analyzing the data presented in Table II, we observed that each project exhibits a distinct context for utilizing draft PRs. However, we observe that the most popular topics are related to: draft for enhancement and improvements that depends on the context and requirement of the project (e.g., UI for web apps like WordPress, Database related changes for the CockroachDB, etc.), experimenting new solutions or ideas, and interestingly carrying out release related activities. For these release-related activities, developers open a draft to share the changes to release and wait for developers' feedback. As stated earlier, feature enhancement depends on the context of the project. Specifically, the Ampproject project predominantly uses draft PRs for feature enhancements (15.2%) or for carrying out release-related activities (37.5%). In contrast, CockroachDB heavily relies on draft PRs for tasks related to database and query optimizations, accounting for a significant majority (52.2%) of their usage since the project is for databases. The Dotnet project, on the other hand, leverages draft PRs for optimizing source code or performing version updates (23.6% and 18.8%, respectively). Remarkably,

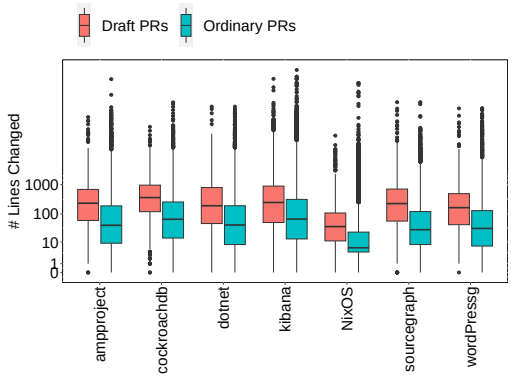


Fig. 3: Number of changed lines for draft PRs

Nixos, which is related to package management, primarily employs draft PRs for dependency management, representing a substantial portion of their utilization (83.5%). In the cases of Kibana and Wordpress, these projects predominantly utilize draft PRs for enhancing the user interface (16.6% for Kibana) or resolving issues associated with their graphical interface (7.6% for Kibana and 42% for Wordpress), reflecting the nature and requirements of these projects. We also observe cases in all the studied projects in which developers explicitly ask for feedback through expressions such as *"this is encouraging .."* and *"I get great results! what do you think?"* to draw their attention and invite input and assistance from others.

Summary of RQ1

The draft mechanism is used by developers to gain feedback on their changes, especially when improving features, releasing new versions, or fixing issues. Drafts are used for larger changes compared to ordinary PRs and are used mostly to gain feedback on changes rather than just simple ideas. **Since the draft mechanism is used for relevant changes, we bring the attention of researchers to focus on such a mechanism for early feedback.**

RQ2. What is the impact of drafts on feedback and integration of changes?

Motivation: We wish to investigate in this research question to which extent the goal of drafts on receiving early comments is achieved. In particular, we investigate the impact of drafts on receiving feedback (issue and review comments) at two different stages of a PR: when the PR is still a draft and when it becomes ready for review. We wish to investigate whether using drafts will have any impact on merging changes. Our intuition is that developers will get early feedback using drafts so their changes can be expected to be of higher quality during the code review and have more chances to be fastly merged.

Approach: To measure the impact of the draft mechanism on the feedback, we quantify the amount of feedback that ordinary PRs and PRs that were drafts receive as well as the time to receive such feedback. In particular, we compare the

Topic Name	Description	ampproject	CockroachDB	Dotnet	Kibana	NixOS	Sourcegraph	Wordpress
Experimentation and Implementation	Testing and validating new ideas in a controlled environment.	0.038	0.071	0.053	0.084	0.006	0.042	0.03
Feature Enhancement and upgrading	Improving and upgrading existing features to enhance functionality and performance.	0.152	0.015	0.006	0.064	0.002	0.045	0.257
Fixing Issues	Identifying, tracking, and resolving bugs.	0.076	0.028	0.035	0.076	0.004	0.118	0.061
Code Source Optimization	Enhancing and optimizing source code.	0.001	0.02	0.236	0.004	0.003	0.01	0.001
Database Optimization	Optimizing database and database queries.	0.013	0.522	0.048	0.025	0.003	0.073	0.014
UI improvement and enhancement	Enhancing and improving the user interface (UI).	0.016	0.017	0.013	0.166	0.0	0.014	0.017
Version Updating	Updating the version for improved features, bug fixes or security enhancements.	0.06	0.035	0.188	0.047	0.076	0.051	0.038
Releasing	Preparing, testing, and distributing new versions.	0.375	0.028	0.062	0.04	0.01	0.057	0.034
Dependency Management	Specifying, resolving, and tracking required libraries, frameworks, or packages.	0.006	0.003	0.005	0.045	0.835	0.004	0.001
Package Updating	Updating the version of some packages.	0.055	0.064	0.161	0.033	0.015	0.078	0.031
Refactoring and Pipeline Improving	Optimizing code structure and development processes.	0.033	0.03	0.035	0.055	0.003	0.196	0.013
Protocol Improvements	Enhancing and optimizing the communication protocols.	0.058	0.083	0.115	0.056	0.032	0.112	0.064
Monitoring Enhancement	enhancements to the collection, analysis, and visualization of metrics, logs, and other relevant data.	0.014	0.03	0.004	0.135	0.002	0.049	0.008
Feature Enhancement	Improving and upgrading existing features to enhance functionality and performance.	0.011	0.036	0.021	0.055	0.006	0.126	0.007
UI Fixes	Fixing visual glitches, addressing layout inconsistencies, enhancing user interactions, improving responsiveness, and ensuring a consistent and intuitive user experience.	0.08	0.002	0.006	0.012	0.002	0.017	0.42
Unkown	Unable to label this topic	0.014	0.016	0.011	0.103	0.003	0.006	0.003

TABLE II: Extracted Topics shared with at least two projects. Note that since a draft can be related to multiple topics, the total percentage for a given project is above 100%.

two types of PRs (i.e, PRs that were drafts vs. ordinary PRs that were not drafts) according to the following metrics:

- Number of comments: we statistically (leveraging Wilcoxon test) compare the number of comments (issue comments and review comments) that draft PRs and PRs that were drafts versus the number of comments that ordinary PRs that were not draft received. Since the goal of drafts is to allow developers to receive comments during the development phase and before a code is ready to review, we investigate which phase (before a change is ready to review and during the review process) do PRs that were drafts receive comments to measure the extent to which the draft mechanism allow developers get early feedback.

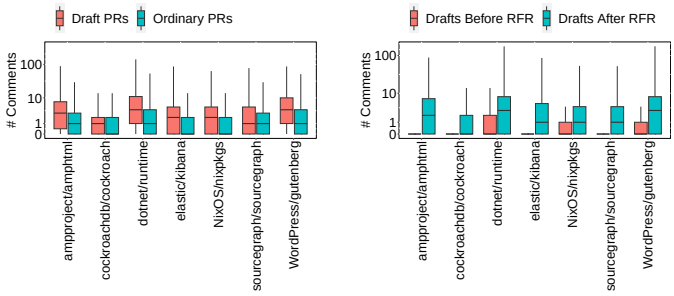
- Number of participants (i.e., these are who review the PR, write comments, and make reactions): We similarly compare draft PRs to ordinary PRs based on the number of participants to see how many people got engaged in discussing draft related changes. Note that we do not consider bots; we only consider human participants in this metric. To identify bots, we leverage the information provided via GitHub Api that indicates whether a comment is from a human or a bot. That said, some accounts are misclassified as human. To avoid such comments, we manually verified around 100 PRs to identify more bots we consider in our list of bots to automatically exclude from the studied comments.

- The time to receive the first comment: We compare the draft and ordinary PR according to the time they receive the first comment. For ordinary PRs (**2.1**), we measure the time between the creation of the PR and the time the first comment was received. For draft PRs, we measure two metrics (**2.2**) the time the first comment was received when a PR is still a draft and (**2.3**) the time the first comment is received after the draft becomes an ordinary PR. Since a PR can take a few minutes to several days to be merged and to reduce the bias caused by the outliers, we normalize the time the first comment is received according to the opening and closing/merge time of the PRs. In particular, we normalize our three metrics as follows:

$$RelativeTimeToFirstCommentPRs = \frac{TimeToFirstComment - PRopen}{PRclose - PRopen} \quad (1)$$

For (2.1), PRopen and PRclose corresponds to the opening and closing time of ordinary PRs. For (2.2), PRclose and PRopen correspond to the time a draft becomes an ordinary PR and the creation time of the draft. For (2.3), PRclose and PRopen correspond to the time of closing the PR and the time a draft becomes ready-for-review (i.e., an ordinary PR).

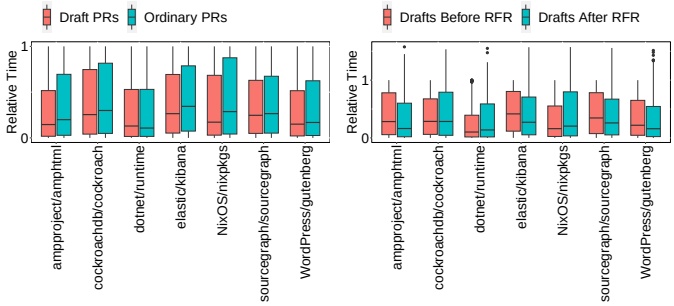
- Merge rate: We investigate whether drafts PRs are more likely to be merged compared to ordinary PRs. The merge rate



(a) #comments for draft and ordinary PRs

(b) #comments for drafts before and after ready-for-review

Fig. 4: #comments that draft and ordinary PRs received



(a) Time to receive the first comment for draft and ordinary PRs

(b) Time to receive the first comment for drafts before and after being ordinary PRs

Fig. 5: Time to receive the first comment

is measured as the number of PRs that end up being merged over the total number of PRs that are merged or closed.

- Time to merge: We investigate whether draft PRs are more likely to be quickly merged compared to ordinary PRs. We define the time to merge an ordinary PR as the time difference between the creation and closing of that PR, whereas the time to merge a PR that was a draft is the time that draft becomes ready for review to the time that PR is merged.

In our comparisons (e.g., the number of comments that drafts PRs received vs. the distribution of comments that ordinary PRs received), we leverage the Wilcoxon rank-sum test with an $\alpha = 0.01$ to compare different distributions.

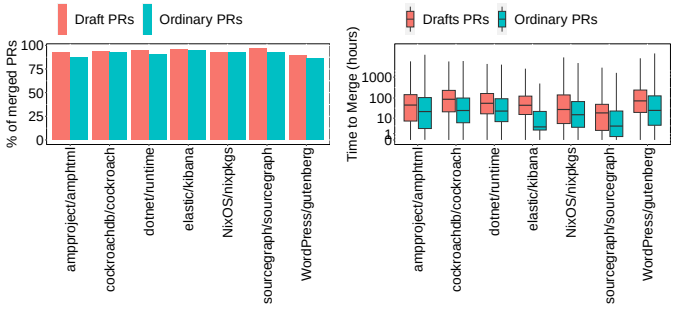
Results: Unfortunately, even if draft PRs receive more comments than ordinary PRs, most of the comments are received after a draft becomes an ordinary PR, as shown in Figure 4. We observe that PRs that were drafts receive a statistically significantly higher (Wilcoxon test; $\alpha = 0.01$) number of comments compared to ordinary PRs for all the seven studied projects, as shown in Figure 4a. In fact, the median number of comments that drafts receive ranges between one to four messages compared to ordinary PRs, which receive a median number of zero to one comment. For example, the median number of comments that the *Dotnet* PRs that were drafts receive is four, while the same median is just one comment for the ordinary PRs of the same project.

However, even if draft PRs receive more comments than ordinary PRs, these comments are added when reviewing the source code change rather than the period on which the PR was a draft, as shown in Figure 4b. Surprisingly, draft PRs receive a median of zero comments when they are still in the draft phase, and when these PRs become ready for review, they receive a median of zero to three comments. That said, the **percentage of PRs that received a comment when they were drafts ranges between 12% to 39.9%**. Some PRs can receive up to 130 comments when they are still drafts. Note that the number of comments received when a PR is still a draft cannot be explained by the time that a PR takes as a draft, as we do not observe any correlation between the number of comments received when a PR is a draft and the amount of time the same PRs take as draft (a Spearman correlation that ranges between 0.32 and 0.49).

While drafts PRs receive more comments, we observe that such a number of comments is not always due to the number of involved participants. In fact, draft PRs involve more participants in a discussion for five out of our seven studied projects. The median number of participants for the PRs which were drafts varies between three to five participants while for ordinary PRs it ranges between two to four different participants. The Spearman correlation between the number of participants and the number of comments is between 0.17 and 0.39. We cannot explain such a larger number of participants engaged in the draft revisions by the amount of changes to review, neither, as the number of participants is not correlated with the code churn (a Spearman correlation that ranges between 0.17 and 0.38).

Draft PRs tend to receive comments at a faster pace after transitioning from the draft phase to being ready for review. Figure 5a reveals that for five out of the seven projects, draft PRs once they become ordinary PRs, tend to attract comments at a considerably faster pace compared to ordinary PR (wilcox test, $\alpha = 0.01$). For Kibana, Ampproject, Sourcegraph, and Wordpress draft PRs after transitioning to the ready for review state they attract comments 0.47 (relative time for draft and ordinary PRs are 0.21 and 0.31), 0.47 (relative time for draft and ordinary PRs are 0.12 and 0.17), 0.26 (relative time for draft and ordinary PRs are 0.21 and 0.27), and 0.32 (relative time for draft and ordinary PRs are 0.12 and 0.16) times more quickly compared to ordinary PRs. Note that the other projects are not statistically significantly different. This can be explained by the fact that draft PRs are often subject to multiple iterations and revisions during the drafting phase, and reviewers are already expecting the upcoming change to review.

While we do not observe a large difference in the merge rate, merging draft PRs takes longer than ordinary PRs. In fact, we observe that the percentage of draft PRs that ended up being merged is slightly higher (between 0.13% to 4.63%) than ordinary PRs, as shown in Figure 6a. We also notice that the draft PRs after becoming ready-for-review take more time to get merged compared to ordinary PRs, as shown in Figure 6b. Draft PRs take a median that ranges between 18.6 and 86.6 hours to be merged, while ordinary PRs take a median of 3.2 to 24.8 hours. Such a finding might be explained by



(a) Merge Rates for draft and ordinary PRs

(b) Merge period of draft and ordinary PRs

Fig. 6: A Comparative Analysis of Merge Rate and Time to Merge for Draft and Ordinary PRs

the amount of changes that are higher for drafts compared to ordinary PRs.

Summary of RQ2

Although PRs that were draft receive more comments than ordinary PRs, drafts do not fully benefit from the advantages they were conceived for as these PRs do not receive many comments when they are still drafts. In addition, the draft mechanism does not improve the integration of code changes, especially the time to merge changes. **Our results suggest that the goal of draft PRs is not fully achieved and there is a need for mechanisms to improve further the mechanism of draft PRs on getting early feedback.**

IV. Analyzing the Main Reasons for Drafts to Get Comments

The goal of this section is to identify to what extent we can leverage machine learning to classify and understand the important factors that distinguish drafts that receive comments from those that do not receive them. Understanding the important factors that lead to comments will help practitioners better understand what they should consider to receive comments, in case they wish to receive them. In particular, we address the following two research questions:

- RQ3. How performant are our classification models in classifying draft PRs?
- RQ4. What are the most important features that distinguish drafts that receive comments from drafts that do not receive comments?

RQ3. How performant are our classification models in classifying draft PRs?

Motivation: The goal of this research question is to evaluate the performance of our classification models and investigate which dimension of metrics significantly influence the likelihood of receiving comments on draft PRs. The goal is to understand the individual and combined impact of various dimensions such as draft features, author activity, and project characteristics on community engagement within the draft process.

	Feature Name	Explanation
PR Features	Description Length	Length of description in words
	Images in Description	Number of images in the description
	Hashtag Present	Indicator if a hashtag is present in the draft
	Tag Present	Indicator if a tag is present in the draft
	Initial Commits	Number of initial commits in the draft
	Initial Additions	Number of added lines in the draft
	Initial Deletions	Number of deleted lines in the draft
	Initial Changed Files	Number of files changed in the draft
Project Features	Project Age	Age of the project at the time of draft creation
	Project Size	Number of contributors at the time of draft creation
	Open Non-Draft PRs	Number of open non-draft PRs
	Open Draft PRs	Number of open draft PRs
	Drafts to Ordinary	Number of drafts that got ready to review
	Merged Draft PRs	Number of merged draft PRs
	Non-Merged Draft PRs	Number of non-merged draft PRs
	Merged Non-Draft PRs	Number of merged non-draft PRs
	Non-Merged Non-Draft PRs	Number of non-merged non-draft PRs
	Integrators Availability	latest activity of the two most active integrators
Author Features	Previous PRs by Author	Number of previous PRs created by the author
	Same Affiliation	Indicates if the author created and integrated prior PRs
	PRs Reviewed by Author	Number of PRs reviewed by the author
	PRs Commented by Author	Number of PRs commented by the author
	Drafts Reviewed by Author	Number of draft PRs reviewed by the author
	Drafts Commented by Author	Number of draft PRs commented by the author
	Previous Drafts by Author	Number of previous drafts created by the author
Events Features	Review Requested	Indicator a reviewer got requested
	Head Ref Force Pushed	Indicator if the draft has head-reference-force-pushed
	Assigned	Indicator if the draft has been assigned
	Unlabeled	Indicator if the draft has been unlabeled
	Review Request Removed	Indicator if a review request is removed
	Labeled	Indicator if the draft is labeled
	Referenced	Indicator if the draft is referenced
	Unassigned	Indicator if the draft is unassigned
	Connected	Indicator if there is connected event
	Mentioned	Indicator if someone got mentioned
	Self-comment	Indicator if the author self (issue/review) comment
	Renamed	Indicator if the draft got renamed

TABLE III: Dimensions of Features

Approach: To classify draft PRs that receive comments from those that do not receive comments, we train machine learning models using the features shown in Table III (inspired from prior work [9]) and leveraging the following pipepline. Note that we distinguish in this RQ and the following review comments from issue comments.

Step 1: Metrics collection: As our goal is to explain the reception of draft pull requests that received a comment during the draft period, we focus our machine learning models on drafts that became ready for review (i.e., ordinary pull requests). To measure the metrics for each draft, we leverage only the drafts and pull requests that existed before the creation of each of our studied drafts. For certain metrics that are related to the review process of prior pull requests, we use only the pull requests that were closed before the creation date of our studied drafts. For example, we only consider the prior pull requests that were closed at the creation time of a studied draft to measure the metrics *Same Affiliation*, *PRs Reviewed by Author*, *PRs commented by Author*, *Drafts Reviewed by Author*, and *Drafts Commented by Author*. Metrics such *Open Non-Draft PRs* and *Open Draft PRs* are calculated based on the PRs and drafts that were created before the creation of each of our studied drafts.

Step 2: Correlation and Redundancy Analysis: As suggested by Jiarpakdee et al. [10], we first eliminate correlated and redundant features, as they can bias the interpretation of our models. To do so, we first leverage the Spearman correlation with a threshold of 0.7. Since correlation does not eliminate all the collinearity, we also remove redundant features. Note that in order to have a consistent interpretation over all the seven studied projects, we try to eliminate the

same features across all the studied projects.

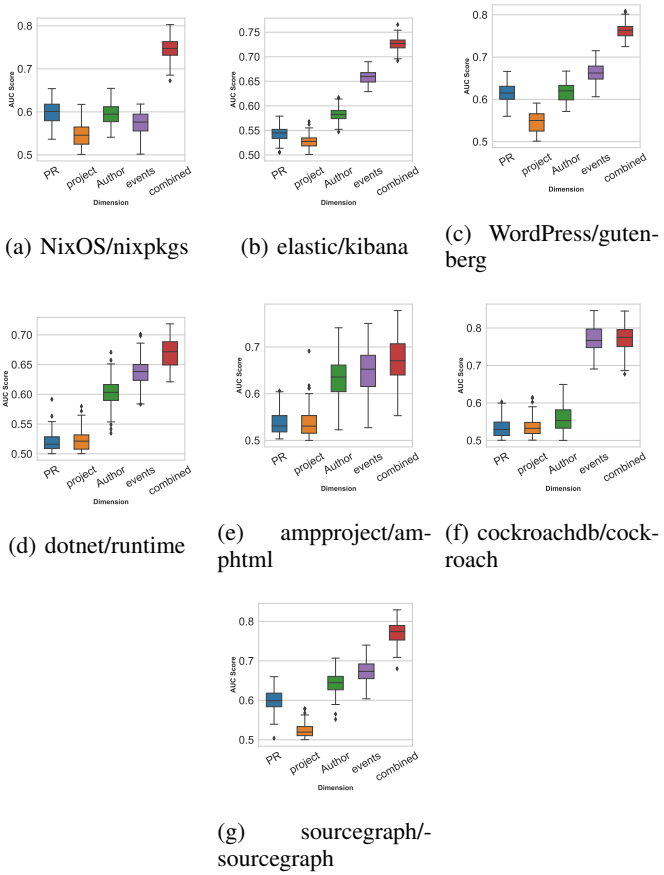
Step 3: Building Models: Since Random Forest have been widely used and showed good performances [11] [12], we also decide to leverage Random Forest models to classify the drafts that received comments from these that do not. Our dependent variable is binary, indicating whether a draft has received a review comment or issue comment. Furthermore, we conduct hyperparameter tuning using grid search. This process involves varying the model parameters (such as the number of trees, depth of the trees, and number of features considered at each split) to find the optimal configuration that maximizes model performance.

Step 4: Evaluate models' performances: To assess the performances of our models, we leverage the 100-out-of-sample-bootstrap technique with replacement, similarly to prior studies [13]. For each bootstrap sample (training dataset), we train the model, as discussed in Step 3, and evaluate that model on the data that were not sampled (testing dataset). The performance of our models is assessed using the Area Under the Receiver Operating Characteristic (AUC-ROC) curve, similarly to prior work [2]. The higher the value of this metric is, the more performant is the model with a minimum AUC value of 0.5, which indicates a random guess. We repeat the same experiment 100 times to end up with 100 AUC values.

Step 5. Assessing the impact of each dimension of features: In addition to the model that is trained on all features, we train models at each of the four dimensions of features (shown in Table III) to assess the contribution of that dimension. The main goal behind this analysis is to assess whether all dimensions contribute to distinguish drafts with comments from drafts without comments. In particular, we train and evaluate (following Steps 3 and 4) a model using just the PR features, another model for Project Features, etc.

Results: Our classification models show promising performances to distinguish drafts with comments from those without comments, whereas all of our studied dimensions show significant explanatory power. As shown in Figure 7 and Figure 8, we first observe that our classification models show classification performances that range between 0.67 and 0.77 for the issue comments and between 0.66 and 0.75 for the review comments. As shown in the same figures, all of our studied dimensions have enough explanatory power (AUC higher than 0.5) to distinguish drafts that received comments from those that did not receive comments. The combination of all dimensions offers greater explanatory power compared to models based on individual dimensions.

What developers take as actions (i.e., events) is the most relevant dimension that distinguishes drafts that receive issue comments from drafts that did not for six out of our seven studied projects, as shown in Figure 7. The Event dimension models outperform all others for five projects, achieving an AUC up to 0.76 across all of our studied projects. The project dimension shows low performance across all seven studied projects, ranking the lowest for five of our seven studied projects. Similarly for the PR dimension, which ranked the lowest for two projects and shows an AUC



(a) NixOS/nixpkgs (b) elastic/kibana (c) WordPress/gutenberg

(d) dotnet/runtime (e) ampproject/amphtml (f) cockroachdb/cockroach

(g) sourcegraph/sourcegraph

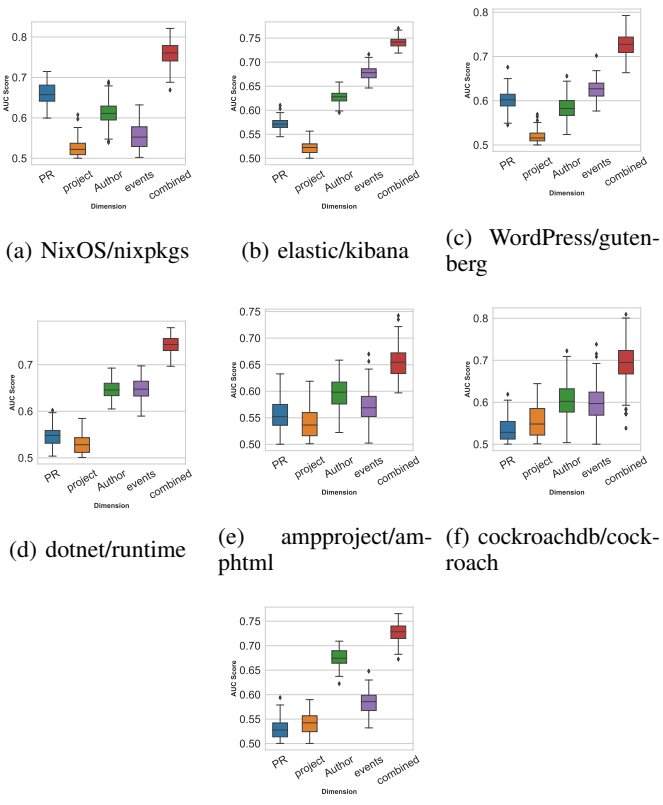
Fig. 7: Evaluation of AUC values across dimensions for issue comments models alongside with the performance of the model that combines all the dimensions.

performance below 0.6 for five projects. That shows that the context, even if it has some expalantory power, is less relevant to distinguish drafts with issue comments from drafts without issue comments. The Author dimension is among to the top-2 dimesions for all the seven projects.

Our results hold for review comments, in which what developers take as actions is more relevant than the context of the draft, as shown in Figure 8. In fact, the Event dimension ranked the best for three out of seven projects and among the top-2 for six projects, while the Project dimension ranked the lowest for five out of seven projects. The remaining dimensions vary from project to project. While the Authors dimension is among the top-2 dimensions for six projects, the PR dimension is among the top-2 dimensions for two projects.

Summary of RQ3

Our models that combine different dimensions show good performances to distinguish drafts that received comments from drafts that did not. In addition, each of our studied dimensions has enough explanatory power that sheds light on the importance of the actions that developers take instead of the global context of the project in which a draft is created.



(a) NixOS/nixpkgs (b) elastic/kibana (c) WordPress/gutenberg

(d) dotnet/runtime (e) ampproject/amphtml (f) cockroachdb/cockroach

(g) sourcegraph/sourcegraph

Fig. 8: Evaluation of AUC values across dimensions for review comments models alongside with the performance of the model that combines all the dimensions.

RQ4. What are the most important features that distinguish drafts that receive comments from drafts that do not receive comments?

Motivation: Since the previous research question shows that our explanatory models have good performances, hence they can be used to explain the differences between drafts that receive comments from those that did not, this research question aims at identifying the most important features for our classification models. These features can provide hints to Github to improve the draft mechanism and hints to developers who wish to receive comments on appropriate actions to take.

Approach: To identify the most important features to our classifiers, we leverage permutation analysis technique similarly to prior studies [14]. It consists of shuffling each feature at a time and measuring the impact of such a shuffle on the AUC performance of the model. The highest the impact, the more important is the feature. We leverage permutation analysis for each model of our 100-bootstrap-based-models to obtain at the 100 ranks, which we classify using the Scott-Knott alogirhtm [15] [16] to obtain a final rank of features from the most important to the least important one. Following the identification of the most important features, we delve deeper into how changes in the values of these features influence the likelihood of a draft PR receiving reviews. To do so, we leverage the Accumulated Local Effects (ALE) plots similarly to prior studies [2]. ALE plots are used to visualize the relationship between a single feature and the outcome predicted by the model. They are particularly adept at highlighting how changes in feature values correlate with variations in the probability of draft PR reviews. Note that all of our ALE figures are in our replication package.

TABLE IV: Top feature ranks across projects for issue comments

	ampproject/amphtml	cockroachdb/cockroach	dotnet/runtime	elastic/kibana	NixOS/nixpkgs	sourcegraph/sourcegraph	WordPress/gutenberg
Count previous drafts by author	**1**	**3**	**1**	**2**	**1**	**2**	**3**
Count prs reviewed by author	**1**	**4**	**2**	**1**	**2**	**1**	**1**
Mentioned	**3**	**2**	**1**	**4**	**5**	**2**	**3**
Integrators Availability	**2**	**3**	**2**	**3**	**3**	**4**	**3**
Open Non Draft PRs	**2**	**3**	**2**	**3**	**3**	**3**	**4**
Description Length	**2**	**3**	**3**	**1**	**2**	**3**	**4**
Initial Additions	**3**	10	**4**	**5**	**2**	**5**	**1**
Head ref force pushed	**4**	**1**	6	9	**4**	6	**5**
Labeled	**5**	**5**	10	**1**	9	6	7
Self issue Comment	**3**	**2**	5	6	6	**2**	**2**
Hashtag Present	**5**	6	7	12	10	6	10
Review requested	**4**	6	7	7	6	7	6
Same affiliation	**4**	8	7	8	5	8	10
Tag present	**5**	10	9	10	6	7	8
Referenced	9	13	12	16	11	11	11
Renamed	6	7	10	10	7	7	8
Review request removed	8	12	12	14	11	10	11
Self review Comment	7	9	10	11	8	9	8
Unlabeled	8	14	11	13	9	12	9
Assigned	7	11	8	10	11	8	7
Connected	9	14	12	15	11	9	11

Results: We observe that mentioning another developer in a draft or requesting a review are among the most important features related to the events for both issue and review comments, as shown in Tables IV and V. As we found in RQ3 that events are relevant, we observe in this research question that mentioning a developer is among the top-five features for all of our seven studied projects to receive an issue comment. We also observe that requesting a review is among the top-five features for five projects and mentioning a developer is among the top-five features for two projects to receive a review comment. We also observe that labeling a draft can help get comments as the event Labeling is ranked among to top-5 features for three and two projects to receive issue and review comments, respectively.

Draft that have detailed descriptions and touch a large amount of code are more likely to receive comments. The description length is indeed among the top-five most important features for both the issue comments and the review comments, as shown in Tables IV and V. According to our analysis of the ALE diagrams, we observe that more detailed descriptions consistently have a higher chance of receiving comments and reviews. We observe that more complex drafts (higher number of additions) are more receptive to reviews,

which contrasts with ordinary PRs, where, as noted in previous studies [17], [18], complex changes are often avoided or criticized, as it is a bad practice. Note that the number of additions is correlated with the number of deletions.

Authors engagement in the project increases the probability of receiving feedback for their draft changes. From our ALE analysis and as shown in Tables IV and V, the more the author has been envolved in previous PRs or drafts, the more likely her draft will receive an issue or review comments. Such a prior engagement can be in the form of commenting or reviewing previous PRs or previous drafts. Our results extend previous work [19], [20] found that the efficiency of the ordinary PRs review process depends on the reciprocity, indicating that developers who actively participate in reviewing others' PRs tend to receive more reviews on their contributions.

The presence and availability of the main integrators significantly influence the reception of comments and reviews for the drafts. We observe that drafts receive more comments or reviews when key integrator are more active in the last three months (feature Integrators Availability), as shown in Tables IV and V. This suggests that the engagement and responsiveness of project leaders or main reviewers play a crucial role in encouraging the review process for draft PRs. This finding is consistent with previous research indicating that active participation in the project can improve the receptiveness of reviews for ordinary PRs [21]. Note that the other features related to the Project dimension are removed from the correlation and redundancy analysis.

TABLE V: Top feature ranks across projects for review comments

	ampproject/amphtml	cockroachdb/cockroach	dotnet/runtime	elastic/kibana	NixOS/nixpkgs	sourcegraph/sourcegraph	WordPress/gutenberg
Count previous drafts by author	**1**	**1**	**1**	**3**	**1**	**1**	**4**
Count prs reviewed by author	**1**	**3**	**3**	**1**	**3**	**1**	**1**
Description Length	**4**	**2**	6	**4**	**3**	**2**	**3**
Initial Additions	**3**	8	**4**	**4**	**3**	**3**	**2**
Head ref force pushed	**5**	**3**	**2**	11	**4**	**4**	**5**
Review requested	**5**	**4**	8	**2**	**5**	**4**	6
Mentioned	**5**	7	8	8	7	11	**5**
Integrators Availability	**4**	**2**	**5**	**5**	**3**	**3**	**3**
Labeled	6	7	13	**3**	10	**4**	8
Open Non Draft PRs	**2**	**3**	7	6	**2**	**3**	**4**
Same affiliation	10	6	10	13	**5**	13	10
Hashtag Present	7	**4**	12	12	13	**5**	12
Self issue comment	9	**5**	9	10	6	10	8
Self review comment	9	6	14	7	9	6	**5**
Referenced	12	10	16	17	12	15	13
Renamed	7	6	11	11	7	8	10
Review request removed	11	9	15	16	11	13	13
Tag Present	8	8	11	12	8	9	9
Unlabeled	12	10	13	14	10	14	11
Assigned	7	7	12	9	13	7	7
Connected	12	10	15	15	14	12	13

Summary of RQ4

Drafts with comments are distinguished from drafts without comments by their longer descriptions and complex changes for which developers who are actively participating in the project and explicitly request a review or mention other developers, taking into account the availability of the main integrators.

V. Implications

We suggest that researchers propose approaches to help GitHub achieve the goal behind the development of the draft mechanism. Since GitHub designed the draft mechanism for developers to receive early comments on their changes or ideas, we have observed that practitioners often do not receive comments or reviews on their drafts, even when the draft mechanism is used for complex changes. We suggest that future research investigate different mechanisms to help practitioners receive comments when they do need them, especially for cases for which a collaboration might be more required such as the preparation of a new version to release or experimenting new ideas or implementations as highlighted in our topic-modeling analysis of RQ1.

We propose practitioners leverage explanatory machine learning models as they show good explanatory performances according to RQ3. Our results show that leveraging different dimensions can help understand what factors can better explain the reception of comments and developers can train models for their projects to better understand factors that increase the chance of getting comments.

We encourage practitioners to take actions to receive comments, especially issue comments. Such actions first start with having a detailed description to their changes, sheding light on the observation that complex changes are more likely to receive comments. Among the actions one can take is to mention other developers to attract their attention or explicitly ask for review when needed.

Following our previous implication, we suggest further studies on assisting practitioners on actions to take to get comments, if needed. As discussed earlier, the length of description is an important factor for receiveing comments, future studies can assist developers writing appropriate descriptions. Similarly, our results suggest taking certain actions such as mentioning a developer to receive comments. Future studies can suggest practitioners in the appropriate actions to take and the appropriate developers to mention to receive comments on drafts.

We encourage developers being active on their projects to increase their chances to receive comments. As we found that commenting or reviewing prior drafts or pull requests increase the chances of receiving comments on her own draft, we recommend practitioners being active.

VI. Related Work

The closest work to our studies focuses on getting feedback on the PR mechanisms (Section VI-A) and how to improve

the code review (Section VI-B).

A. Studies on Understanding Feedback in PRs:

Several studies (e.g., [2], [22]–[26]) focused on improving the mechanisms of the PRs for better collaboration. Zhang et al. [22] conducted an exploratory study on the impact of @-mention in pull-request. Hu et al. [23] tackled the challenge of providing adequate feedback to software developers and software students in their study. Chopra et al. [24] explored the impact of referencing behaviours in PR discussions, focusing on how referencing patterns influence collaborative software development. Batoun et al. [25] investigated reactions, as a source of feedback, and provided guidelines to researchers on how to use reactions. Another factor related to receiving timely feedback on the success of a PR was examined by Hasan et al. [26]. Their study revealed that bots can play a major role in reducing the time it takes for a PR to receive its first response, but they can also contribute to delays if they are not configured properly. Their findings were confirmed by Khatoonabadi et al. [2], who found that not properly used bots could lead developers to abandon PRs.

Another line of research has extensively examined the role of comments in GitHub PRs (e.g., [27] [28] [29], [30]). For example, Tsay et al. [28] specifically investigated how social and technical factors, including comments, influence the evaluation of contributions in GitHub. Meanwhile, Yu et al. [30] and Ying et al. [29] explored the use of PR comments as a feature for recommending reviewers. Middleton et al. [27] took a broader approach and employed PR comments, along with other types of contributions, to predict the long-term involvement of new developers in a software project.

While those prior works investigated understanding the feedback exchange in PRs, none of them examined the impact of using draft PRs on receiving comments.

B. Studies on Enhancing Code Review Feedback in PRs:

Several studies (e.g., [31] [17] [32] [33] [34]) explored the code reviews in the context of PRs. For example, Gousios et al. [31] looked into how technical factors that impact the acceptance of PRs and the time it takes to accept them. Gousios et al. [17] confirmed that the maintainers' decision to accept a PR is primarily based on its quality, with particular emphasis on its adherence to the project's style and architecture, code quality, and test coverage. Recently, Wessel et al. [32] investigated the adoption of code review bots in Open Source Software (OSS) projects and their impact on group dynamics. In the pursuit of automating the code review process, Zhiyu Li et al. [33] focused on utilizing pre-training techniques and proposed CodeReviewer model, a pre-trained model tailored for code review scenarios. Additionally, Hong et al. [34] addressed the labor-intensive nature of manual code review by proposing CommentFinder, a retrieval-based approach that recommends code review comments.

As shown in the work mentioned above, examining the code review feedback on the success of a PR to be accepted is of critical importance in particular, for software maintainers. Hence, our work examines the draft PR mechanism and its impact on providing feedback to contributors.

VII. Threats to Validity

External threat to validity: Our external threat to validity is related to the generalizability of our results to other software systems. while, our study consider seven case studies that are popular open source projects and we found consistent results and conclusions over these four studies in our analysis, we encourage future work to replicate our study on other systems. We do not generalise our results to ordinary pull requests, while we encourage future studies replicate our model on the chances of ordinary pull requests to receive comments.

Internal threat to validity: A first internal threat to validity is related to the number of topics that are discussed in the draft PRs. While we manually choose the such a number, using another number of topics might allow the identification of other topics that are not covered in our analysis. To mitigate this risk, two authors reviewed the obtained topics and adjust that number until reaching cohesive topics.

VIII. Conclusion

The draft PR mechanism allows developers to receive comments at an early stage of development, and they can get comments on their ideas before implementing them or on ongoing development. While a large body of research focused on the PR mechanism and how to improve it, no prior studies looked at the draft mechanism, which is the gap we wish to fill in this paper. In particular, we empirically identify how practitioners use the PRs, the impact of the PRs on feedback (receiving comments in particular), and what are the important factors related to the obtention of feedback.

We found in our studied seven projects that the draft mechanism is mostly used for larger changes, which hints at the complexity of changes that are associated with the draft PRs. We mainly observe different purposes for using the draft mechanism that can be regrouped into three major themes: enhancing existing code, experimenting with new ideas, and preparing a release. Most of the uses of drafts are around the implementation of new ideas or features, while a few drafts are used without any source code modifications hence to discuss just ideas. The draft mechanism gets more comments than ordinary PRs but these comments are received after marking a draft as ready for review hence becoming an ordinary PR. Thus, our result shows that the goal of the draft PRs on getting feedback is not achieved.

We leverage explanatory machine learning models to further understand the most important factors that distinguish drafts with comments from drafts without comments (i.e., issue comment or review comment). We found that our models reach good performances, from 0.67 to 0.77 and 0.66 to 0.75 for issue comments and review comments, respectively. The interpretation of our models shows that the most important features are related to the actions developers take, the availability of integrators, the description of the drafts, and the active involvement in the projects.

REFERENCES

[1] Github-Blogs, "Introducing draft pull requests — the github blog." https://github.blog/2019-02-14-introducing-draft-pull-requests/. (Accessed on 03/23/2022).

[2] S. Khatoonabadi, D. E. Costa, R. Abdalkareem, and E. Shihab, "On wasted contributions: Understanding the dynamics of contributor-abandoned pull requests–a mixed-methods study of 10 large open-source projects," *ACM Transactions on Software Engineering and Methodology*, vol. 32, pp. 1–39, jan 2023.

[3] G. Zhao, D. A. Costa, and Y. Zou, "Improving the pull requests review process using learning-to-rank algorithms," *Empirical Softw. Engg.*, vol. 24, p. 2140–2170, aug 2019.

[4] 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.

[5] A. K. McCallum, "Mallet: A machine learning for language toolkit." http://mallet.cs.umass.edu, 2002.

[6] L. Heng, C. T.-H. P., S. Weiyi, and H. A. E, "Studying software logging using topic models," *Empirical Software Engineering*, pp. 2655–2694, 2018.

[7] C. Tse-Hsun, T. S. W, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering*, pp. 1843–1919, 2016.

[8] B. Anton, T. S. W, and H. A. E, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, pp. 619–654, 2014.

[9] K. Hasan, M. Macedo, Y. Tian, B. Adams, and S. Ding, "Understanding the time to first response in github pull requests," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, (Los Alamitos, CA, USA), pp. 1–11, IEEE Computer Society, may 2023.

[10] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 320–331, 2021.

[11] T. R. Benala and K. Tantati, "Efficiency of oversampling methods for enhancing software defect prediction by using imbalanced data," *Innov. Syst. Softw. Eng.*, vol. 19, p. 247–263, jun 2022.

[12] M. S. Rahman, E. Rivera, F. Khomh, Y.-G. Guéhéneuc, and B. Lehnert, "Machine learning software engineering in practice: An industrial case study," 2019.

[13] D. Lee, G. K. Rajbahadur, D. Lin, M. Sayagh, C.-P. Bezemer, and A. E. Hassan, "An empirical study of the characteristics of popular minecraft mods," *Empirical Software Engineering*, vol. 25, pp. 3396–3429, aug 2020.

[14] J. Jiarpakdee, C. K. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 166–185, 2022.

[15] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," no. 1, 2017.

[16] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization for defect prediction models," 2018.

[17] G. Gousios, A. Zaidman, M.-A. Storey, and A. v. Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 358–368, 2015.

[18] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 1028–1038, 2016.

[19] M. El Mezouar, F. Zhang, and Y. Zou, "An empirical study on the teams structures in social coding using github projects," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3790–3823, 2019.

[20] M. M. Rahman, C. K. Roy, and J. A. Collins, "Correct: Code reviewer recommendation in github based on cross-project and technology experience," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 222–231, 2016.

[21] Z. Liao, Y. Li, D. He, J. Wu, Y. Zhang, and X. Fan, "Topic-based integrator matching for pull request," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, p. 1–6, IEEE Press, 2017.

[22] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "A exploratory study of @-mention in github's pull-requests," vol. 1, (Jeju Island, Korea, Republic of), pp. 343 – 350, 2014. At-mention;Collaborative software development;Current situation;Exploratory studies;Pull-request;Social development;Social media;Social media tools;.

[23] Z. Hu and E. Gehringer, "Use bots to improve github pull-request feedback," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, (New York, NY, USA), p. 1262–1263, Association for Computing Machinery, 2019.

[24] A. Chopra, M. Mo, S. Dodson, I. Beschastnikh, S. S. Fels, and D. Yoon, ""@alex, this fixes 9": Analysis of referencing patterns in pull request discussions," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, oct 2021.

[25] M. A. Batoun, K. L. Yung, Y. Tian, and M. Sayagh, "An empirical study on github pull requests' reactions," *ACM Trans. Softw. Eng. Methodol.*, may 2023. Just Accepted.

[26] K. A. Hasan, M. Macedo, Y. Tian, B. Adams, and S. Ding, "Understanding the time to first response in github pull requests," 2023.

[27] J. Middleton, E. Murphy-Hill, D. Green, A. Meade, R. Mayer, D. White, and S. McDonald, "Which contributions predict whether developers are accepted into github teams," in *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, (New York, NY, USA), p. 403–413, Association for Computing Machinery, 2018.

[28] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), p. 356–366, Association for Computing Machinery, 2014.

[29] H. Ying, L. Chen, T. Liang, and J. Wu, "Earec: Leveraging expertise and authority for pull-request reviewer recommendation in github," in *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*, CSI-SE '16, (New York, NY, USA), p. 29–35, Association for Computing Machinery, 2016.

[30] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1, pp. 335–342, 2014.

[31] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), p. 345–355, Association for Computing Machinery, 2014.

[32] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Quality gatekeepers: investigating the effects of code review bots on pull request activities," *Empirical Software Engineering*, vol. 27, may 2022.

[33] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, and N. Sundaresan, "Automating code review activities by large-scale pre-training," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, (New York, NY, USA), p. 1035–1047, Association for Computing Machinery, 2022.

[34] Y. Hong, C. Tantithamthavorn, P. Thongtanunam, and A. Aleti, "Commentfinder: A simpler, faster, more accurate code review comments recommendation," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, (New York, NY, USA), p. 507–519, Association for Computing Machinery, 2022.