Pull Request Abandonment in Open-Source Projects

SayedHassan Khatoonabadi

A Thesis

In the Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Computer Science)

at Concordia University

Montréal, Québec, Canada

November 2023

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: SayedHassan Khatoonabadi

Entitled: Pull Request Abandonment in Open-Source Projects

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Chair
Dr. Sang Hyeok Han

External Examiner
Dr. Andy Zaidman

Examiner
Dr. Ferhat Khendek

Examiner
Dr. Nikolaos Tsantalis

Examiner
Dr. Tse-Hsun (Peter) Chen

Thesis Supervisor (s)
Dr. Emad Shihab

Approved by

Dr. Leila Kosseim Chair of Department or Graduate Program Director

January 19, 2024
Date of Defence

Dr. Mourad Debbabi Dean, Gina Cody School of Engineering and Computer Science

Abstract

Pull Request Abandonment in Open-Source Projects

SayedHassan Khatoonabadi, Ph.D.

Concordia University, 2023

Pull-based development is a common paradigm for contributing to and reviewing code changes in numerous open-source projects. However, a considerable amount of Pull Requests (PRs) with valid contributions are not finalized because their contributors have left the review process unfinished. Such abandoned PRs waste a considerable amount of time and effort from both their contributors and their maintainers. Furthermore, PRs that are neither progressed nor resolved, clutter the list of PRs, and eventually make it difficult for the maintainers to manage and prioritize unresolved PRs. Recognizing these challenges, this thesis aims to investigate the underlying dynamics of abandoned PRs, evaluate the helpfulness of common solutions to PR abandonment, and propose ways to mitigate PR abandonment in large open-source projects. We start by studying the characteristics of abandoned PRs, the reasons why contributors abandon their PRs, and the perspectives of project maintainers on dealing with PR abandonment. Our findings indicate that contributors and the review process play a more prominent role in PR abandonment than projects and PRs themselves. Our survey with project maintainers also indicates that Stale bot is commonly adopted by many open-source projects to deal with abandoned PRs. However, there are ongoing debates on whether using Stale bot alleviates or exacerbates PR abandonment. Therefore, in our next study, we investigate the reliance of projects on Stale bot to deal with their PR backlog, the impact of Stale bot on pull-based development, and the kind of PRs usually intervened by Stale bot. Our findings indicate that despite its benefits, Stale bot tends to further aggravate contributor abandonment. To help better mitigate PR abandonment, in our last study, we propose a machine learning approach to predict the first response latency of the maintainers and the contributor of a PR. We demonstrate the effectiveness of our approach in both project-specific and cross-project settings and also discuss the importance and impact of different features on the predicted waiting times. The awareness fostered by these predictions enables both the maintainers and the contributor to take proactive actions to mitigate potential challenges during the review process of the PR before it gets abandoned.

Acknowledgments

This journey of pursuing my Ph.D. has been a profound experience, filled with challenges, learning, and growth. It would not have been possible without the support and guidance of many individuals to whom I owe my deepest gratitude.

First and foremost, I extend my sincerest thanks to my supervisor, Dr. Emad Shihab, for his continued support, encouragement, and mentorship. His expertise, patience, and insightful feedback have been instrumental in shaping my research and academic development.

I am deeply grateful to my committee members, Dr. Andy Zaidman, Dr. Nikolaos Tsantalis, Dr. Tse-Hsun (Peter) Chen, and Dr. Ferhat Khendek. Their constructive criticism and valuable comments have significantly contributed to the depth and quality of my research.

I would also like to acknowledge the collaboration and mentorship of our ex-postdocs, Dr. Diego Costa and Dr. Ahmad Abdellatif. Working alongside them has been an enriching experience, contributing significantly to my academic growth and the quality of our joint publications.

A special thanks goes to my colleagues at DAS Lab. The discussions and collaborative spirit we shared have been a constant source of motivation. The lab has been more than just a workplace; it has been a community of friends who have supported each other through both academic and personal challenges.

Lastly, I must express my profound gratitude to my dear family and friends for their love, understanding, and unwavering belief in my abilities. Their constant encouragement and support have been my strength throughout this journey.

This thesis is not only a reflection of my work but also a testament to the collective effort and support of all these remarkable individuals.

Dedication

To *Reihan*, my beloved wife, whose profound love and unwavering companionship have been my refuge in both challenging and joyous times. Your enduring strength and support are the heartbeat of my every achievement.

To *Effat*, my dear mother, whose love and devotion have not only shaped this achievement but also the individual I am today. Your continual belief in my abilities has been a guiding force, lighting my way through every obstacle and success on this journey.

And in loving memory of *Reza*, my father, whose cherished memories remain a source of inspiration in my life. I deeply wish he could have witnessed this moment; his absence is felt with every milestone I reach. He is forever in my heart, unforgettable, and dearly missed.

Their unconditional love, sacrifices, and faith in me have been the cornerstones of my success, making this journey not just possible, but also meaningful.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Research Statement

1.1 Introduction

Pull-based development has become a common paradigm for contributing to and reviewing code changes in numerous open-source projects [1, 2]. Pull Requests (PRs) are the driving force behind the maintenance and evolution of these projects, encompassing everything from bug fixes to new features. This development model offers higher information centralization, process automation, and tool integration, which facilitates communication and increases awareness during the code review process [3, 1]. This in turn reduces the time and effort required to contribute, review, improve, and integrate code changes [1, 2]. Contributors initiate this collaborative process by submitting a PR that proposes changes for integration into the project. The PR then undergoes a review process, during which the contributor revises the changes based on feedback from the project maintainers. This cycle repeats until the PR satisfies the maintainers' requirements for getting merged [4, 5].

The streamlined contribution mechanism enabled by PRs has encouraged numerous developers to contribute to open-source projects with fewer barriers [1, 6, 2]. However, a considerable amount of PRs with valid contributions are not finalized because their contributors have left the review process unfinished [7, 8]. Unfortunately, abandoned PRs waste a considerable amount of time and effort that is often put in both by the contributors to prepare and submit such PRs and by the maintainers to manage and review them [8]. Furthermore, PRs that are neither progressed nor resolved, accumulate over time, clutter the list of PRs, and eventually make it difficult for the maintainers to manage and prioritize unresolved PRs [8, 9]. As a real-world example, a large backlog of unresolved PRs led the DefinitelyTyped project to declare "bankruptcy" in June

2016. Consequently, they closed all unresolved PRs submitted before May 2016 just to be able to start afresh [10].

The great opportunity cost and wasted efforts arising from PR abandonment make it a non-trivial challenge for the open-source community. To mitigate this challenge, we first need to better understand the underlying dynamics of abandoned PRs. Therefore, we start by investigating the characteristics of abandoned PRs, the reasons why contributors abandon their PRs, and the perspectives of project maintainers on dealing with PR abandonment. Our findings indicate that most PRs are abandoned because of the obstacles faced by the contributors and the hurdles imposed by the maintainers during the review process. We also find that contributors and the review process play a more prominent role in PR abandonment than projects and PRs themselves. Additionally, our survey with project maintainers indicates that Stale bot [11] is commonly adopted by many open-source projects on GitHub to automatically triage inactive and abandoned PRs [12, 13]. However, there are ongoing debates on whether using Stale bot alleviates or exacerbates the problem of inactive and abandoned PRs [14, 15, 16, 17, 18, 19, 20]. Therefore, in our next study, we aim to better understand the helpfulness of Stale bot for pull-based development. Specifically, we investigate the reliance of projects on Stale bot to deal with their PR backlog, the impact of Stale bot on pull-based development, and the kind of PRs usually intervened by Stale bot. Our findings indicate that despite its helpfulness in dealing with a backlog of unresolved PRs, Stale bot tends to decrease community engagement and increase the risk of contributor abandonment.

However, to help better mitigate PR abandonment, we need to acknowledge the significant role of the responsiveness of the maintainers and the contributor during the review process of a PR as indicated by the findings of our first study. The first responses are of particular importance as they not only directly influence the duration [21, 22] and the outcome [23, 24, 8] of the review process, but also the likelihood of future contributions by the contributor [25, 21]. Therefore, we propose a machine learning approach to predict: (1) the first response latency of the *maintainers* following the submission of a PR, and (2) the first response latency of the *contributor* after receiving the first response from the maintainers. We believe that by predicting the first response latencies, our approach helps open-source projects facilitate collaboration between maintainers and contributors during the review process of PRs. Contributors, when aware of anticipated waiting times, can adjust their schedules accordingly, reducing uncertainty and preserving their motivation throughout the review process [24, 26]. Maintainers, aware of possible delays in contributor responses, can proactively offer additional support or take action to mitigate potential blockers [24, 9]. This awareness also allows maintainers

to better allocate their time and resources and prioritize PR reviews [9]. Furthermore, analyzing response time trends can help projects pinpoint and rectify bottlenecks, thereby enhancing the efficiency and effectiveness of their PR review workflows.

1.2 Research Statement

Motivated by the challenges of abandoned PRs, the goal of this PhD thesis is to explore the reasons and the solutions to PR abandonment. We state our research statement as follows:

> PR abandonment is a non-trivial challenge that results in a significant opportunity cost for the open-source community. We investigate the underlying dynamics of abandoned PRs, evaluate the helpfulness of common solutions to PR abandonment, and propose ways to mitigate PR abandonment in large open-source projects.

1.3 Thesis Overview

In this section, we provide an overview of the work presented in this thesis and highlight the main results of each work.

Chapter 3: Understanding the Dynamics of Contributor-Abandoned Pull Requests

Pull-based development has enabled numerous volunteers to contribute to open-source projects with fewer barriers. Nevertheless, a considerable amount of PRs with valid contributions are abandoned by their contributors, wasting the effort and time put in by both the contributors and maintainers. To better understand the underlying dynamics of contributor-abandoned PRs, we conduct a mixed-methods study using both quantitative and qualitative methods. We curate a dataset consisting of 265,325 PRs that include 4,450 abandoned ones from 10 large and popular open-source projects on GitHub and extract 16 features characterizing PRs, contributors, review processes, and projects.

First, we analyze the characteristics of abandoned PRs to understand which features of PRs, their contributors, their review processes, and their projects are associated with PR abandonment. We find that abandoned PRs are usually more complex, their contributors are usually less experienced, and their review process is usually lengthier than nonabandoned PRs. We also observe that the project features play both a

positive and negative role in PR abandonment, where abandoned PRs have become more frequent in three projects and less frequent in five other projects as the projects mature. Then, we rely on machine learning techniques to determine the relative importance of the features and describe how each feature varies the predicted abandonment probability of PRs. We find that the features of review processes, contributors, and projects are more important for predicting PR abandonment than the features of PRs themselves. Specifically, PRs with more than three responses from either the participants or the contributors, and those submitted by novice contributors are more likely to get abandoned. Also, the abandonment probability changes as the projects evolve, with half the projects showing a decrease in abandonment in their mature stages and the other half showing an increase in abandonment.

Next, we manually examine a random sample of abandoned PRs to identify the probable reasons why contributors abandon their PRs. We find that difficulty addressing the maintainers' comments, lack of timely review from the maintainers, difficulty resolving the CI failures, and difficulty resolving the merge issues are the most frequent reasons why contributors abandon their PRs. Afterward, we survey the top core maintainers of the projects to gain additional insights on how they deal with or suggest dealing with abandoned PRs and their perspectives on our findings. We find that Stale bot is commonly used to automatically deal with abandoned PRs. However, the maintainers recommend establishing a triage mechanism to help find problems and support PRs that need assistance before they become abandoned. Finally, we combine the findings from our research questions and survey responses to discuss the role of PRs, contributors, review processes, and projects in PR abandonment. Our findings indicate that contributors and the review process play a more prominent role in PR abandonment than projects and PRs themselves. This work has been published in the *ACM Transactions on Software Engineering and Methodology* journal [24].

Chapter 4: Understanding the Helpfulness of Stale Bot for Pull-based Development

PRs that are neither progressed nor resolved clutter the list of PRs, making it difficult for the maintainers to manage and prioritize unresolved PRs. To automatically track, follow up, and close such inactive PRs, Stale bot was introduced by GitHub. Despite its increasing adoption, there are ongoing debates on whether using Stale bot alleviates or exacerbates the problem of inactive PRs. To better understand the helpfulness of Stale bot for pull-based development, we conduct a quantitative empirical study. We curate a dataset of 20 large and popular open-source projects on GitHub and measure 13 performance indicators covering resolved PRs, review latency, resolution time, review discussion, PR updates, and contributor retention. We also extract 16

factors characterizing PRs, contributors, and review processes.

First, we analyze the configuration and activity of Stale bot to understand the extent to which the projects rely on Stale bot to automatically deal with their unresolved PRs. We find that the usage level of Stale bot widely varies among the projects. On average each month, Stale bot intervened in less than 25% of open PRs in nine projects, between 25% and 50% of open PRs in five projects, and more than 50% of open PRs in six projects. Then, we apply interrupted time-series analysis [27] as a well-established quasi-experiment to understand if and how adopting Stale bot improves the efficiency and effectiveness of the pull-based development workflow. We find that the projects closed more PRs within the first few months of adopting Stale bot, but overall closed and merged fewer PRs afterward. The adoption of Stale bot is also associated with faster first reviews in merged PRs, faster resolutions in closed PRs, slightly fewer updates in merged PRs, and considerably fewer active contributors in the projects.

Next, we analyze the characteristics of PRs intervened by Stale bot, as well as their contributors and review processes, to understand the factors that are associated with a higher probability of getting intervened by Stale bot. We find that Stale bot tends to intervene more in complex PRs, PRs from novice contributors, and PRs with lengthy review processes. Specifically, besides the resolution time of PRs, the largest differences are observed in the number of prior PRs by contributors, the mean response latency of PRs, the acceptance rate of contributors, and the contribution period of contributors. Finally, we combine our findings to discuss the advantages and disadvantages of adopting Stale bot for pull-based development. Our findings indicate that Stale bot can help projects deal with a backlog of unresolved PRs and also improve the review process of PRs. However, Stale bot can also negatively affect the contributors (especially novice or casual contributors) of projects. Therefore, relying solely on Stale bot to deal with inactive PRs may lead to decreased community engagement and an increased probability of contributor abandonment. This work has been published in the *ACM Transactions on Software Engineering and Methodology* journal [28].

Chapter 5: Predicting the First Response Latency of Maintainers and Contributors in Pull Requests

The success of a PR depends on the responsiveness of the maintainers and the contributor during the review process. Being aware of the expected waiting times can lead to better interactions and managed expectations for both the maintainers and the contributor. Therefore, we propose a machine learning approach

to predict the first response latency of the maintainers following the submission of a PR, and the first response latency of the contributor after receiving the first response from the maintainers. We curate a dataset of 20 large and popular open-source projects on GitHub and extract 21 features to characterize projects, contributors, PRs, and review processes.

First, we evaluate seven types of classifiers to identify the best-performing models. We find that our best-performing models achieve an average improvement of 29% in AUC-ROC and 51% in AUC-PR for maintainers, as well as 39% in AUC-ROC and 89% in AUC-PR for contributors compared to a no-skilled classifier across the projects. Then, we conduct permutation feature importance and SHAP analyses to understand the importance and impact of different features on the predicted response latencies. We find that PRs submitted earlier in the week, containing an average or slightly above-average number of commits, and with concise descriptions are more likely to receive faster first responses from the maintainers. Similarly, PRs with a lower first response latency from maintainers, that received the first response of maintainers earlier in the week, and containing an average or slightly above-average number of commits tend to receive faster first responses from the contributors. Additionally, contributors with a higher acceptance rate and a history of timely responses in the project are likely to both obtain and provide faster first responses.

Finally, we evaluate our approach in a cross-project setting, where the models achieve an average improvement of 33% in AUC-ROC and 58% in AUC-PR for maintainers, as well as an average improvement of 42% in AUC-ROC and 95% in AUC-PR for contributors. Furthermore, we find that the key predictors in the cross-project setting are: submission day, number of commits, contributor acceptance rate, historical maintainers responsiveness, and historical contributor responsiveness for maintainers' first response latency; and first review latency, review day, historical contributor responsiveness, number of commits, and contributor activity within the PR for contributors' first response latency. We believe that the awareness fostered by these predictions not only leads to managed expectations for both the maintainers and the contributor of a PR but also enables them to take proactive actions to mitigate potential challenges during the review process before the PR gets abandoned. This work has been submitted to the *IEEE Transactions on Software Engineering* journal for review.

1.4 Thesis Contributions

The main contributions of this thesis are as follows:

- We provide empirical evidence on the characteristics of abandoned PRs, the reasons why contributors abandon their PRs, and the perspectives of project maintainers on dealing with PR abandonment.
- We provide empirical evidence on the reliance of projects on Stale bot to deal with their PR backlog, the impact of Stale bot on pull-based development, and the kind of PRs usually intervened by Stale bot.
- We propose machine learning models for predicting the first response latency of maintainers and contributors in PRs and discuss the importance and impact of different factors on the anticipated waiting periods.
- We publicly share our scripts and datasets online to promote the reproducibility of our research and facilitate future work related to this thesis.

1.5 Related Publications

The work presented in this thesis has been published in or submitted to the following venues:

- **SayedHassan Khatoonabadi**, Diego Elias Costa, Rabe Abdalkareem, and Emad 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*, 32(1):1–39, 2023. DOI:10.1145/3530785.
- **SayedHassan Khatoonabadi**, Diego Elias Costa, Suhaib Mujahid, and Emad Shihab. Understanding the helpfulness of Stale bot for pull-based development: an empirical study of 20 large open-source projects. *ACM Transactions on Software Engineering and Methodology*, early access, 2023. DOI:10.1145/3624739.
- **SayedHassan Khatoonabadi**, Ahmad Abdellatif, Diego Elias Costa, and Emad Shihab. Predicting the first response latency of maintainers and contributors in pull requests. *IEEE Transactions on Software Engineering*, under review, 2023. DOI:10.48550/arXiv.2311.07786.

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews the related work to our thesis. Then, Chapters 3 and 4 presents our studies on understanding the dynamics of contributor-abandoned PRs and the

helpfulness of Stale bot for pull-based development. Next, Chapter 5 provides our approach for predicting the first response latency of maintainers and contributors in PRs. Finally, Chapter 6 summarizes the thesis and discusses the key directions for future work.

Chapter 2

Background and Related Work

In this chapter, we provide an overview of the relevant work to our thesis. First, we review studies on the reasons and consequences of PR abandonment. Then, we review studies on the usage and challenges of Stale bot before moving on to the studies on the impact of tools on pull-based development. Finally, we provide a summary of studies on the review latency and decision of PRs, followed by an overview of studies on duplicated PRs and redundant changes. At the end of each section, we also explain how each work in our thesis contributes to the body of knowledge.

2.1 Reasons and Consequences of PR Abandonment

In pull-based development, developers fork a project (i.e., create a personal copy of the project) before making their changes. Whenever ready, the developers request their changes to get merged into the project by submitting a PR. The PR then undergoes a review process, during which the contributor revises the changes based on feedback from the project maintainers. This cycle repeats until the PR satisfies the maintainers' requirements for getting merged [4, 5]. However, an industrial report [7] estimates that 8% of PRs are wasted and never merged. In contrast to rejected PRs, abandoned PRs are valid contributions that are not finalized because their contributors have left the review process unfinished. Unfortunately, abandoned PRs waste a considerable amount of time and effort that is often put in both by the contributors to prepare and submit such PRs and by the maintainers to manage and review them. However, PR abandonment as a challenge that results in a great opportunity cost for the open-source community has only recently received attention from Li et al. [8]. They manually examined 321 abandoned PRs from five open-source projects on GitHub (namely,

Cocos2d-x, Kubernetes, Node.js, Rails, and Rust) to identify the reasons why PRs get abandoned by their contributors, the impact of abandoned PRs on the maintainers, and the strategies adopted by the projects to deal with abandoned PRs. Then, they quantified the frequency of the identified reasons, impacts, and strategies by surveying 710 developers. They observed that PRs are abandoned mostly due to the lack of maintainers' responsiveness and the lack of contributors' time and interest. They also reported that PR abandonment increases the efforts needed to manage and maintain projects because abandoned PRs clutter the list of PRs, waste review efforts, require additional attention for a careful close, delay landing of interdependent PRs, result in duplicated PRs, disorder project milestones, and finally leave a bad impression.

While this study discussed the developers' perspective on PR abandonment, the influence of the factors related to PRs, contributors, review processes, and projects on the abandonment probability of PRs is still not known. To fill this knowledge gap, in Chapter 3, we conduct a mixed-methods study using both quantitative and qualitative methods on a larger dataset consisting of 10 large open-source projects on GitHub. Specifically, we investigate the characteristics of abandoned PRs, the reasons why contributors abandon their PRs, and the perspectives of project maintainers on dealing with PR abandonment.

2.2 Usage and Challenges of Stale bot

Stale bot was released in 2017 to automatically triage abandoned issues and PRs and is adopted by many open-source projects on GitHub [11, 13, 12]. However, little is known about the usage and challenges of Stale bot in the literature. In a preliminary study, Wessel et al. [12] investigated how projects adapt and maintain Stale bot over time by analyzing its configuration history in 765 open-source projects. They found that most projects use Stale bot to triage both their issues and PRs. Furthermore, they found that while most projects do not modify the configuration of Stale bot after its initial setup, the few that do rarely make more than three modifications subsequently. Therefore, they concluded that setting up and using Stale bot does not require much effort from the projects. Several studies [16, 17, 18, 19, 20] have also incidentally mentioned that Stale bot introduces noise and friction for both the contributors and the maintainers.

Nevertheless, if and how Stale bot can actually be helpful to open-source projects has not yet been studied. To fill this knowledge gap, in Chapter 4, we conduct a quantitative empirical study to understand the helpfulness of Stale bot in the context of pull-based development. Specifically, we investigate the reliance of projects on Stale bot to deal with their PR backlog, the impact of Stale bot on pull-based development, and

the kind of PRs usually intervened by Stale bot.

2.3 Impact of Tools on Pull-based Development

Open-source projects are adopting various automation tools and bots to improve the efficiency and effectiveness of their pull-based development workflow [29, 30, 31]. Several studies have evaluated the impact of adopting such tools in open-source projects by applying interrupted time-series analysis [27] (a.k.a. regression discontinuity design). Zhao et al. [32] was the first to apply this method for understanding the impact of adopting Travis CI. They found that the adoption slows down the increasing trend in the number of merge commits, accelerates the decreasing trend in the merge commit churn, and reverses the increasing trend in the number of closed PRs. Cassee et al. [33] further studied the impact of adopting Travis CI and found that it also slows down the increasing trend in the number of discussion comments in PRs.

Wessel et al. [34] studied the impact of adopting code review bots and found that it accelerates the increasing trend in the number of merged PRs, accelerates the decreasing trend in the number of closed PRs, decreases the number of comments in all PRs, increases the resolution time of merged PRs, and reverses the increasing trend in the resolution time of closed PRs. In another study, Wessel et al. [35] investigated the impact of adopting GitHub Actions and found that it decreases the number of both merged PRs and closed PRs, increases the number of discussion comments in merged PRs, decreases the number of discussion comments in closed PRs, increases the resolution time of merged PRs, and decreases the resolution time of closed PRs.

However, our study in Chapter 4 focuses on investigating the impact of adopting Stale bot to better understand its helpfulness for pull-based development.

2.4 Review Latency and Decision of PRs

The literature has extensively studied how various technical, social, and personal factors influence the review latency and decision of PRs. Gousios et al. [1] was the first to investigate how technical factors affect the merge decision and merge time of PRs. They found that the merge decision is mainly affected by whether the PR touches recently modified code, while the merge time is affected by the track record of the developer, as well as the size of the project, its test coverage, and its openness to external contributions. Gousios et al. [9]

reports that the decision of integrators to accept a contribution is based on its quality, especially conformance to the project style and architecture, source code quality, and test coverage.

Tsay et al. [36] showed that in addition to technical factors, social factors could influence the acceptance of PRs. They found that while PRs with lots of comments are associated with a lower probability of getting accepted, the prior interactions of submitters in the project moderate this effect. Additionally, they found that well-established projects are more conservative while evaluating contributions. Soares et al. [37] found that the programming language, the number of commits, and the number of files added in a PR, as well as whether its contributor is an external developer and whether it is the contributor's first PR, influence the merge decision and merge time. Yu et al. [38] found that projects prefer PRs that are small, have less controversy, and are submitted by trusted contributors.

Kononenko et al. [39] found that the size of PRs, the number of participants in the review discussions, and the contributor's experience and affiliation influence both the review time and merge decision. Moreover, they reported that developers consider PR quality, type of change, and responsiveness of the contributor as important factors in the merge decision. Developers perceive the quality of a PR by its description, complexity, and revertability; and the quality of a review by its feedback, tests, and discussions. Pinto et al. [40] found that in company-owned open-source projects, external contributors compared to the employees face significantly more rejections and have to wait longer to receive a decision on their contributions. Zou et al. [41] found that PRs that violate the code style of projects are more likely to get rejected and take longer to get closed. Lenarduzzi et al. [42] found that code quality does not affect the acceptance of PRs, and suggested that other factors such as the maintainer's reputation and the feature's importance might be more influential on PR acceptance.

Several studies have also investigated how demographic characteristics of contributors can influence the outcome of PRs. Terrell et al. [43] found that among external contributors, women whose gender is identifiable have lower acceptance rates. Rastogi [44] and Rastogi et al. [45] also found that PRs are more likely to get accepted when both the contributors and the maintainers are from the same geographical location. Moreover, Nadri et al. [46] found evidence of bias against perceptible non-White races. Later, Nadri et al. [47] found that contributions from perceptible White developers have a higher acceptance chance, and perceptible non-White contributors are more likely to get their PRs accepted if the maintainer is also from the same race. Furtado et al. [48] also found that contributors from countries with low human development indexes submit fewer PRs but face the most rejections.

Besides social and technical factors, Iyer et al. [49] also studied how personality traits [50] influence the decision of PRs. They found that personal and technical factors play a significant and comparable role in PR acceptance, but still not to the extent of social factors. Additionally, they observed that contributors who are more open and conscientious but less extroverted have a higher chance of getting their PRs accepted. Similarly, maintainers who are more conscientious, extroverted, and neurotic accept more PRs. Recently, Zhang et al. [22] conducted a large-scale empirical study to understand how a range of factors, identified through a systematic literature review, can explain the latency of PRs under different scenarios. They found that the description length is the most influential factor when PRs are submitted. When closing PRs, using CI tools, or when the contributor and the integrator differ, the presence of comments is the most influential factor. When comments are present, the latency of the first comment is the most influential.

Zhang et al. [23] also conducted a similar comprehensive empirical study to investigate how a range of factors, identified through a systematic literature review, can explain the decision of PRs under different scenarios. Most notably, they found that the area hotness of PR is influential only in the early stage of project development and becomes less influential as projects mature.

Hasan et al. [21] is the first to conduct an exploratory study on the time-to-first-response of PRs in GitHub. They found that first responses in PRs are often generated by bots. They also observed that complex PRs with lengthy descriptions and inexperienced contributors with less communicative attitudes tend to experience longer delays in receiving the first human response. Kudrjavets et al. [51] reported the waiting time from the proposal of code changes until the first response as nonproductive time that negatively affects code velocity.

While these studies focus on understanding the review latency and outcome of PRs, our work in Chapter 5 specifically predicts the first response latency of maintainers following the submission of a PR. Besides, we also predict the first response latency of the contributor of a PR after receiving the first response from the maintainers.

2.5 Duplicated PRs and Redundant Changes

Li et al. [52] found that duplicate PRs waste human and computing resources and adversely impact OSS development. To facilitate studies on duplicated PRs, Yu et al. [53] compiled a dataset of duplicated PRs from 26 popular GitHub projects. To identify duplicate PRs, Li et al. [54] proposed an approach that uses textual information within PRs to automatically identify similar PRs. Li et al. [55] extended the previous

work by also considering the change information of PRs. Ren et al. [56] proposed an approach to identify redundant code changes in forks as early as possible. Wang et al. [57] enhanced the performance of the previous approach by considering the time factor.

While abandoned PRs can lead to duplicated PRs, this thesis explores the reasons and the solutions to PR abandonment.

2.6 Chapter Summary

PR abandonment as a challenge that results in a great opportunity cost for the open-source community has only recently received attention from Li et al. [8]. While their findings shed light on PR abandonment from the perspective of developers, the influence of the factors related to PRs, contributors, review processes, and projects on PR abandonment is still unknown. To gain a more comprehensive understanding of the underlying dynamics of contributor-abandoned PRs, we conduct a mixed-methods study in Chapter 3. As part of this study, we found that Stale bot is commonly used to deal with inactive and abandoned PRs. However, the helpfulness of Stale bot has not yet been empirically validated. Therefore, in Chapter 4, we conduct a quantitative study to better understand if and how adopting Stale bot helps open-source projects in their pull-based development workflow. This study revealed that Stale bot tends to further aggravate contributor abandonment. Therefore, in Chapter 5 we explore an alternative approach to better mitigate PR abandonment. Our approach predicts the first response latency of maintainers and contributors of PRs that enable them to take proactive actions to mitigate potential challenges during the review process of the PR before it gets overdue and eventually abandoned.

Chapter 3

Understanding the Dynamics of Contributor-Abandoned Pull Requests

3.1 Introduction

Pull-based development has been popularized by social coding platforms such as GitHub and is widely adopted by distributed software teams, especially within the open-source community [1]. In this development model, developers fork a project (i.e., create a personal copy of the project) before making their changes. Whenever ready, the developers request their changes to get merged into the project by submitting a PR. The maintainers then review the PR and decide whether to merge it into their project. Compared to traditional methods, pull-based development reduces the time taken to review and merge the contributions [1, 2].

The streamlined contribution mechanism enabled by PRs has encouraged numerous external developers to contribute to open-source projects with fewer barriers [1, 6, 2]. However, an industrial report [7] estimates that 8% of PRs are wasted and never merged. Such PRs are either rejected by the maintainers or abandoned by their contributors. In contrast to rejected PRs, abandoned PRs are valid contributions that are not finalized because their contributors have left the review process unfinished. Unfortunately, abandoned PRs waste a considerable amount of time and effort that is often put in both by the contributors to prepare and submit such PRs and by the maintainers to manage and review them.

The literature has extensively studied how various technical, social, and personal factors influence the acceptance and review process of PRs. However, PR abandonment as a challenge that results in a great

opportunity cost for the open-source community, especially for the contributors and the reviewers of abandoned PRs, has only recently received attention from Li et al. [8]. Based on a survey of open-source developers, they explained how abandoned PRs impact project maintainers and discussed why developers abandon their PRs. While their findings shed light on PR abandonment from the perspective of developers, the influence of the factors related to PRs, contributors, review processes, and projects on PR abandonment is still unknown.

To gain a better and more comprehensive understanding of the underlying dynamics of contributor-abandoned PRs, we conduct a mixed-methods study using both quantitative and qualitative methods [58]. *For the sake of brevity, we refer to external contributors as contributors throughout this chapter.* First, we curate a dataset consisting of 265,325 contributor PRs from 10 popular and mature GitHub projects (namely, Homebrew Cask, Kubernetes, Kibana, Ansible, DefinitelyTyped, Rust, Odoo, Legacy Homebrew, Elasticsearch, and Swift). Then, we devise heuristics to identify 4,450 candidate PRs with a high chance of being truly abandoned by their contributors. Next, we measure 16 features to characterize the PRs, their contributors, their review processes, and their projects for our quantitative analyses. We aim to answer the following three research questions in this chapter:

RQ1: What are the significant features of contributor-abandoned PRs in the studied projects? We find that contributor-abandoned PRs are usually more complex, their contributors are usually less experienced, and their review process is usually lengthier than nonabandoned PRs. Furthermore, as the projects mature, contributor-abandoned PRs have become more frequent in three projects (i.e., Kubernetes, Swift, and DefinitelyTyped) and less frequent in five other projects (i.e., Kibana, Ansible, Elasticsearch, Odoo, and Homebrew Cask).

RQ2: How do different features impact the probability of PR abandonment in the studied projects? We find that the features of the review process, contributor, and project are more important in predicting PR abandonment than the features of PRs themselves. Specifically, PRs with more than three responses from the participants or the contributors, and those submitted by novice contributors are more likely to get abandoned. Also, the abandonment probability changes as the projects evolve, with half of the projects showing a decrease in abandonment in their mature stages and the other half showing an increase in abandonment.

RQ3: What are the probable reasons why contributors abandon their PRs in the studied projects? We find that the most frequent abandonment reasons are related to the obstacles faced by contributors

followed by the hurdles imposed by maintainers during the review process. Specifically, difficulty addressing the maintainers' comments, lack of review from the maintainers, difficulty resolving the CI failures, and difficulty resolving the merge issues are the most common reasons why contributors abandon their PRs.

3.1.1 Our Contributions

In summary, we make the following key contributions in this chapter:

- We identify the features of PRs, their contributors, their review processes, and their projects that significantly differ between abandoned and nonabandoned PRs.
- We rank the features based on their relative importance for predicting PR abandonment and describe how different values of these features vary the predicted probability of abandonment.
- We identify the probable reasons why contributors abandon their PRs and survey the core developers of studied projects to understand their perspectives on dealing with PR abandonment and our findings.
- To promote the reproducibility of our study and facilitate future research, we also share our dataset at https://doi.org/10.5281/zenodo.4892276.

3.1.2 Chapter Organization

The remainder of this chapter is organized as follows. Section 3.2 presents our research methodology and Sections 3.3 to 3.5 present our findings for each research question. Then, Section 3.6 reports the perspectives of maintainers on dealing with PR abandonment and our findings. Next, Section 3.7 further discusses our findings. Finally, Section 3.8 discusses the limitations of our study and Section 3.9 concludes this chapter.

3.2 Methodology

In the following, we explain how we design our study (Section 3.2.1), select the study projects (Section 3.2.2), collect the required data (Section 3.2.3), identify abandoned PRs (Section 3.2.4), and extract features from PRs (Section 3.2.5).

Table 3.1: Overview of the projects selected to study contributor-abandoned PRs.

Project	PRs	Stars	Contributors	Months	Domain	Language(s)
Homebrew Cask	78,446	17,077	7,246	98	Package Manager	Ruby
Kubernetes	56,721	66,644	3,628	71	Container Orchestration	Go
Kibana	43,324	14,313	896	87	Analytics Dashboard	TypeScript
Ansible	42,338	43,333	7,168	98	Automation Platform	Python
DefinitelyTyped	38,645	28,316	13,866	91	Type Definitions	TypeScript
Rust	38,361	45,261	3,181	116	Programming Language	Rust
Odoo	38,241	17,636	1,822	72	Business Apps	JavaScript, Python
Legacy Homebrew	33,577	27,786	7,904	75	Package Manager	Ruby
Elasticsearch	33,411	49,134	2,350	116	Analytics Engine	Java
Swift	31,984	51,831	974	54	Programming Language	C++, Swift

3.2.1 Study Design

To gain a more comprehensive understanding of the underlying dynamics of contributor-abandoned PRs, we conduct a mixed-methods study using both quantitative and qualitative methods [58]. First, we perform statistical analysis to identify the significant features of abandoned PRs in RQ1 (Section 3.3). Then, we use machine learning techniques to determine the relative importance of the features and describe how each feature varies the predicted probability of abandonment in RQ2 (Section 3.4). Finally, we manually examine a random sample of 4,450 abandoned PRs to identify the reasons why contributors abandon their PRs in RQ3 (Section 3.5).

3.2.2 Studied Projects

For our study, we need open-source projects that are popular among the community and have a rich history of adopting pull-based development. For this purpose, we rely on GitHub as a pioneer in supporting the PR model and the largest open-source ecosystem [59], which have also been the subject of many software engineering studies [60]. To focus on the most popular projects, we use the number of stars as a proxy [61] and retrieve the list of the top 1,000 most-starred projects. Among these projects, we focus on the top 10 with the most number of PRs to ensure that each project has enough historical data for our study. As shown in Table 3.1, the studied projects cover multiple application domains and programming languages, with each project having at least 14 thousand stars, 31 thousand PRs, 800 external contributors, and 4 years of PR history.

3.2.3 Data Collection

To identify abandoned PRs, we require the timeline activity of PRs, which records all the events during the lifecycle of a PR. For this purpose, we use the `PyGithub` package [62] to retrieve the required data from GitHub. On May 30th, 2020, we collected the timeline, commits, and changed files metadata [63, 64] for the 435,048 PRs of the studied projects.

3.2.4 Abandoned PRs Identification

After collecting the PRs data, we need to identify those abandoned by their contributors. Each PR in GitHub has one of the following three states: (i) *open* indicates that the PR is not finalized and might be in progress, (ii) *closed* indicates that the PR is either rejected by the maintainers or abandoned by its contributor, and (iii) *merged* indicates that the PR is merged into the project. Abandoned PRs are a subset of the *open* or *closed* PRs that are wasted because their contributors have left them unfinished. The contributors of such PRs may either explicitly declare their abandonment decision or implicitly stop addressing the maintainers' comments. The maintainers often employ bots like Stale [11] to close abandoned PRs after a period of inactivity [12], or they manually find and close the abandoned PRs.

However, GitHub does not assign a specific status for abandoned PRs to explicitly distinguish them from nonabandoned ones. Therefore, we cannot simply retrieve the list of abandoned PRs neither directly through the GitHub API [65] nor using existing archives such as GHTorrent [66] and GH Archive [67]. Therefore, we resort to heuristics to identify abandoned PRs based on the collected metadata. Heuristics are not guaranteed to be optimal and are subject to an inherent tradeoff between their accuracy and completeness [68]. To determine the best balance between the precision and recall of our dataset, we experimented with different heuristics before finalizing the following:

Step 1: Exclude PRs from core developers. Our study focuses on contributions from external developers, which are more prone to get abandoned. Therefore, we exclude the PRs from core developers to focus on external contributions. GitHub defines different roles for the authors of PRs within a project [69]. Among these roles, *owner* refers to the owners of the project, *member* refers to the members of the organization owning the project, and *collaborator* refers to those invited to collaborate on the project. Since these three roles typically have push/merge permissions within a GitHub repository, we consider them as core developers and exclude their PRs from our dataset.

Step 2: Exclude PRs from deleted accounts. GitHub allows its users to remove their accounts permanently and afterward refers to the contributors of PRs from such accounts as *ghost*. Since there is no straightforward way to distinguish between the contributors of such PRs [60], we exclude them from our study.

Step 3: Exclude recently updated PRs. To minimize the chance of marking PRs that are still in progress as abandoned, we exclude the PRs that their contributors have recently updated. To be conservative, we exclude the PRs that their contributors have updated (i.e., new comments or commits) within the last six months of the data collection date (i.e., May 30th, 2020).

Step 4: Exclude merged PRs. We consider merged PRs as not wasted and thus not abandoned. However, the maintainers may not always use the merging methods provided through the GitHub interface to merge PRs. To account for such PRs, we resort to heuristics similar to Kalliamvakou et al. [60]. Specifically, we exclude the PRs with a *merged* status (i.e., merged using the GitHub interface) and those PRs that are closed without a *merged* status, but have a merged commit inside the project that references them (e.g., "Close #123").

Step 5: Search keywords in discussion comments. As the last step for identifying abandoned PRs, we rely on keyword searching within all the discussion comments of PRs similar to Li et al. [8]. First, we remove code snippets and reply quotes from these comments and then search for keywords representing the unresponsiveness of contributors. To determine such keywords, we consider the keywords used in Li et al. [8] as our initial set. Then, we manually examine a sample of known abandoned PRs from our studied projects and iteratively refine our keywords. Finally, we find the following keywords are commonly used to refer to abandoned PRs:

{*abandon, stale, any update, lack of update, no update, inactive, inactivity, lack of activity, no activity, not active, lack of reply, no reply, lack of response, and no response*}.

Using our heuristics, we identified 4,450 abandoned PRs among the curated 265,325 PRs. As with any heuristic, ours may return some nonabandoned PRs (i.e., false positives), given that the review process of PRs often involves social interactions between the contributors and the reviewers before getting finalized. To validate the quality of our dataset, we manually analyze 100 PRs (10 PRs from each project) to verify if they have been truly marked as abandoned. We find seven false positives out of the 100 examined PRs as these PRs were rejected while including the keywords representing abandonment (e.g., [70]). Still, we believe that a false-positive rate of 7% gives us enough confidence to rely on this dataset for our study.

Table 3.2: Overview of the features extracted to characterize PRs, their contributors, their review process, and their projects.

Dimension	Feature	Description
Pull Request	pr_description	Number of words in the title and description of the PR
	pr_commits	Number of commits during the lifecycle of the PR
	pr_changed_lines	Number of changed lines during the lifecycle of the PR
	pr_changed_files	Number of changed files during the lifecycle of the PR
Contributor	contributor_contribution_period	Number of months since the first PR of the contributor in the project
	contributor_pulls	Number of prior PRs by the contributor in the project
	contributor_acceptance_rate	Ratio of previously merged PRs by the contributor in the project
	contributor_abandonment_rate	Ratio of previously abandoned PRs by the contributor in the project
Review Process	review_response_latency	Number of days till the first response in the PR
	review_participants	Number of participants during the lifecycle of the PR
	review_participants_responses	Number of responses by the participants during the lifecycle of the PR
	review_contributor_responses	Number of responses by the contributor during the lifecycle of the PR
Project	project_age	Number of months since the starting date of the project
	project_pulls	Number of prior PRs in the project
	project_contributors	Number of prior contributors in the project
	project_open_pulls	Number of open PRs in the project at the submission time of the PR

3.2.5 Feature Extraction

To identify the features that are possibly associated with PR abandonment, we consult the literature on pull-based development [71, 72, 22, 23]. As shown in Table 3.2, we extract 16 features covering four different dimensions: (i) PR features, (ii) contributor features, (iii) review process features, and (iv) project features. In the following, we describe the extracted features for each dimension in more detail.

PR Features:

Description Length. The description length of PRs is found to negatively impact their acceptance probability and review time [38]. We aim to understand whether PRs with shorter descriptions are more frequently abandoned than verbosely described PRs. To characterize a PR's description length, we measure the number of words that have been used in its title and description (denoted by *pr_description*).

Change Complexity. The complexity of changes has been extensively shown to negatively impact the acceptance probability and the review time of PRs [36, 37, 38, 39]. We aim to understand whether complex PRs are more prone to get abandoned. To characterize a PR's change complexity, we measure the number of commits that have been submitted during the PR's lifecycle (denoted by *pr_commits*); the number of lines (denoted by *pr_changed_lines*); and the number of files (denoted by *pr_changed_files*) that have been changed

(i.e., additions or deletions) as part of the submitted commits.

Contributor Features:

Experience Level. The experience of contributors has been extensively shown to positively impact the acceptance probability and the review time of PRs [1, 37, 38, 39]. We aim to understand whether more experienced contributors are less likely to abandon their PRs. To characterize a contributor's experience within a project, we measure the number of months that have been elapsed since the first submitted PR of the contributor to the project (denoted by *contributor_contribution_period*); the number of PRs that the contributor has previously submitted to the project (denoted by *contributor_pulls*); and the ratio of the previously submitted PRs by the contributor that had been merged into the project (denoted by *contributor_acceptance_rate*).

Abandonment History. To the best of our knowledge, the abandonment history of contributors has not been previously studied. We aim to understand whether contributors who have a long history of abandonment are more likely to abandon their PRs. To characterize a contributor's abandonment history within a project, we measure the ratio of the previously submitted PRs by the contributor that we have marked as abandoned in the project (denoted by *contributor_abandonment_rate*).

Review Process Features:

Response Latency. The response latency is found to negatively impact the acceptance probability and the review time of PRs [38]. We aim to understand whether PRs that take longer to receive a first response from the reviewers are more likely to get abandoned. To characterize a PR's response latency, we measure the number of days that have been taken to receive their first response (i.e., comment or review) from the participants (denoted by *review_response_latency*).

Participants Activity. The activity of participants (i.e., anyone participating in the review process except the contributor) is found to negatively impact the acceptance probability of PRs [36, 39]. We aim to understand whether PRs with a higher activity from their participants are more likely to get abandoned. To characterize the participants' activity in a PR, we measure the number of participants in its review process (denoted by *review_participants*); and the number of responses (i.e., comments or reviews) that have been submitted by the participants (denoted by *review_participants_responses*) during the review process.

Contributor Activity. Similar to the participants' activity, we aim to understand whether PRs with a

higher activity from their contributors are also more likely to get abandoned. To characterize the contributor's activity in a PR, we measure the number of responses (i.e., comments or self-reviews) that the contributor has submitted during the review process of the PR (denoted by *review_contributor_responses*).

Project Features:

Maturity Level. The maturity of projects is found to have a mixed impact on the acceptance probability and the review time of their PRs [36, 38]. We aim to understand whether the rate of abandoned PRs changes as projects become more mature. To characterize a project's maturity, we measure the number of months that have been elapsed since the creation date of the project until the submission date of the PR (denoted by *project_age*); the number of PRs that have been previously submitted to the project (denoted by *project_pulls*); and the number of developers who have previously contributed to the project (denoted by *project_contributors*) at the submission time of the PR.

Maintainers Workload. The workload of maintainers is found to negatively impact the acceptance probability and the review time of PRs [38]. We aim to understand whether the high workload of maintainers increases the rate of abandoned PRs. To characterize a project's workload, we measure the number of submitted PRs that were still open at the submission time of the PR (denoted by *project_open_pulls*).

3.3 RQ1: What are the significant features of contributor-abandoned PRs in the studied projects?

PR abandonment is a challenge that results in a significant opportunity cost for the open-source community, especially for the contributors and the reviewers of abandoned PRs. A recent study by Li et al. [8] has surveyed open-source developers to explain why PRs become abandoned. However, the influence of different factors on PR abandonment has not been studied yet. As our first research question, we aim to understand which features of PRs, their contributors, their review processes, and their projects are associated with PR abandonment. Specifically, we want to investigate how significantly abandoned PRs differ from nonabandoned ones.

3.3.1 Approach

We perform statistical analyses to identify the significant features of abandoned PRs compared with nonabandoned PRs. First, we compare the distribution of the extracted features between abandoned and

nonabandoned PRs and then test their statistical and practical significance. In the following, we explain each step in more detail:

Step 1: Compare distribution of features. To compare the distribution of features between abandoned and nonabandoned PRs, we generate violin plots [73] for each project using the `ggstatsplot` package [74]. The generated plots for each feature are presented in Section 6.2.5, specifying their median values (denoted by M), interquartile ranges (the box inside the violin), and probability densities (the width of the violin at each value).

Step 2: Test statistical significance of features. To test the statistical difference between the features of abandoned and nonabandoned PRs, we apply the MannWhitney U test [75] with a 95% confidence level (i.e., $\alpha = 0.05$). We use this nonparametric test because we cannot assume the distribution of our features to be normal. To calculate this statistic, we use the `stats` package [76] and add the results to the plots generated in Step 1. For easier comparison, we denote $p < 0.05$ with *, $p < 0.01$ with **, and $p < 0.001$ with ***.

Step 3: Test practical significance of features. While statistical significance verifies whether a difference exists between the features of abandoned and nonabandoned PRs, we also need to test their practical difference [77]. For this purpose, we use Cliff's delta [78] to estimate their magnitude of difference (i.e., effect size). The value of Cliff's delta (denoted by d) ranges from -1 to $+1$: a positive d implies that the values of the feature in abandoned PRs are often greater than those of nonabandoned PRs, while a negative d implies the opposite. To calculate this statistic, we use the `effectsize` package [79] and add the results to the plots generated in Step 1. For easier comparison, we convert the d values to qualitative magnitudes based on the following thresholds as suggested by Hess and Kromrey [80]:

$$\text{Effect size} = \begin{cases} \text{Negligible}, & \text{if } |d| \leq 0.147 \\ \text{Small}, & \text{if } 0.147 < |d| \leq 0.33 \\ \text{Medium}, & \text{if } 0.33 < |d| \leq 0.474 \\ \text{Large}, & \text{if } 0.474 < |d| \leq 1 \end{cases}$$

3.3.2 Findings

Table 3.3 summarizes the significance of different features across the studied projects. We consider a feature significant if its difference between abandoned and nonabandoned PRs is both statistically significant

Table 3.3: Significance of different features across the studied projects.

Dimension	Feature	Significant	Small	Medium	Large
Pull Request	pr_description	8	7 (↑)	1 (↑)	–
	pr_commits	6	6 (↑)	–	–
	pr_changed_lines	3	1 (↑)	–	2 (↑)
	pr_changed_files	–	–	–	–
Contributor	contributor_pulls	**10**	4 (↓)	3 (↓)	3 (↓)
	contributor_acceptance_rate	**10**	5 (↓)	4 (↓)	1 (↓)
	contributor_contribution_period	5	2 (↓)	1 (↓)	2 (↓)
	contributor_abandonment_rate	2	2 (↑)	–	–
Review Process	review_participants_responses	**10**	1 (↑)	1 (↑)	8 (↑)
	review_participants	**10**	2 (↑)	1 (↑)	7 (↑)
	review_contributor_responses	7	3 (↑)	3 (↑)	1 (↑)
	review_response_latency	4	4 (↑)	–	–
Project	project_age	8	6 (↑↓)	–	2 (↑↓)
	project_pulls	8	6 (↑↓)	–	2 (↑↓)
	project_contributors	8	6 (↑↓)	–	2 (↑↓)
	project_open_pulls	6	4 (↑↓)	–	2 (↑↓)

↑ shows that abandoned PRs have values greater than nonabandoned ones, ↓ shows that abandoned PRs have values smaller than nonabandoned ones, and ↑↓ shows a mixed relationship.

(i.e., $p < 0.05$) and practically significant (i.e., the effect size is small, medium, or large) in at least one project. Overall, we observe that the most significant features are related to the review process and contributors of PRs. We also find that four features (characterizing the review process and contributor) are significant across all the projects, and eight other features (encompassing all the dimensions) are significant in at least half the projects. In the following, we discuss the significance of each dimension in more detail.

Abandoned PRs are usually more complex than nonabandoned PRs. As shown in the PR dimension of Table 3.3, abandoned PRs tend to have lengthier descriptions (8 projects), contain more commits (6 projects), and involve more changed lines (3 projects). However, abandoned and nonabandoned PRs tend to be similar in their number of changed files across all the projects. The results suggest that abandoned PRs receive even more effort from their contributors, highlighting the waste resulting from the abandonment.

The contributors of abandoned PRs usually have less experience than the contributors of nonabandoned PRs. As shown in the contributor dimension of Table 3.3, the contributors of abandoned PRs tend to have previously submitted fewer PRs (all the projects), have a lower acceptance rate (all the projects), have a lower contribution period (5 projects), and have a higher abandonment rate (2 projects). However, the

results cannot be attributed to the expected higher familiarity and expertise of the maintainers because we only consider external contributors in our study (see Section 3.2.4).

The review process of abandoned PRs is usually lengthier than the review process of nonabandoned PRs. As shown in the review process dimension of Table 3.3, the review process of abandoned PRs tends to receive more responses from its participants (all the projects), involve more participants (all the projects), receive more responses from the contributors (7 projects), and have a higher latency to receive the first response from the participants (4 projects). The results suggest that abandoned PRs are not just abandoned after the PR was submitted but have received even more effort from both their contributors and reviewer, again highlighting the waste resulting from the abandonment.

The project features play both a positive and negative role in PR abandonment. As shown in the project dimension of Table 3.3, we observe contrasting patterns in how the rate of abandoned PRs change alongside the project maturity (8 projects) or workload (6 projects). For easier comparison, we group these projects based on their similarities (i.e., positive or negative) in Table 3.4. In the first group (i.e., Kubernetes, Swift, and DefinitelyTyped), abandoned PRs tend to become more frequent as the projects become more mature (i.e., an increase in project_age, project_pulls, or project_contributors). In two of these projects (i.e., Kubernetes and Swift), abandoned PRs also become more frequent as they experience a higher workload (i.e., an increase in project_open_pulls). In contrast to the first group, the second group (i.e., Kibana, Ansible, Elasticsearch, Odoo, and Homebrew Cask) experienced fewer abandoned PRs as the projects become more mature (i.e., an increase in project_age, project_pulls, or project_contributors). Surprisingly, in four of these projects (i.e., Kibana, Ansible, Elasticsearch, and Odoo), abandoned PRs are more frequent when the projects have a lower workload (i.e., a decrease in project_open_pulls). The results may be associated with the change in the team structure, policies, or processes. For example, DefinitelyTyped has refined its review process since 2016 by rotatively assigning a TypeScript employee each week to focus on merging PRs [81].

Answer to RQ1. Our findings suggest that contributor-abandoned PRs are usually more complex, their contributors are usually less experienced, and their review process is usually lengthier than nonabandoned PRs. Furthermore, as the projects mature, contributor-abandoned PRs have become more frequent in three projects (i.e., Kubernetes, Swift, and DefinitelyTyped) and less frequent in five other projects (i.e., Kibana, Ansible, Elasticsearch, Odoo, and Homebrew Cask).

Table 3.4: Difference of the project features between abandoned and nonabandoned PRs.

Group	Project	Maturity			Workload
		project_age	**project_pulls**	**project_contributors**	**project_open_pulls**
I	Kubernetes	Large (↑)	Large (↑)	Large (↑)	Large (↑)
	Swift	Small (↑)	Small (↑)	Small (↑)	Small (↑)
	DefinitelyTyped	Small (↑)	Small (↑)	Small (↑)	–
II	Kibana	Large (↓)	Large (↓)	Large (↓)	Large (↓)
	Ansible	Small (↓)	Small (↓)	Small (↓)	Small (↓)
	Elasticsearch	Small (↓)	Small (↓)	Small (↓)	Small (↓)
	Odoo	Small (↓)	Small (↓)	Small (↓)	Small (↓)
	Homebrew Cask	Small (↓)	Small (↓)	Small (↓)	–

3.4 RQ2: How do different features impact the probability of PR abandonment in the studied projects?

In RQ1, we investigated what features of PRs, their contributors, their review processes, and their projects are associated with PR abandonment. As our second research question, we aim to better understand which PRs have a higher probability of getting abandoned by their contributors. Specifically, we want to identify which features are the most important for predicting PR abandonment and describe how each feature can influence the abandonment probability of PRs.

3.4.1 Approach

We use machine learning techniques to understand how each feature varies the predicted probability of PRs getting abandoned. First, we consider the features that we found to be significant in abandoned PRs and remove correlated and redundant features to ensure the quality of our models. Then, we build and evaluate the classifier models that we later use to analyze the relative importance and impact of each feature on the abandonment probability. In the following, we explain each step in more detail:

Step 1: Remove insignificant features. In RQ1, we found that the number of changed files in a PR (i.e., *pr_changed_files*) does not significantly differ between abandoned and nonabandoned PRs in any of the studied projects. Therefore, we exclude this feature because it is not valuable for analyzing the abandonment probability of PRs and consider the remaining 15 features for our analysis.

Step 2: Remove correlated features. To focus on the most important features, we eliminate highly correlated features, which negatively affect the interpretation of models [82]. To check the monotonic

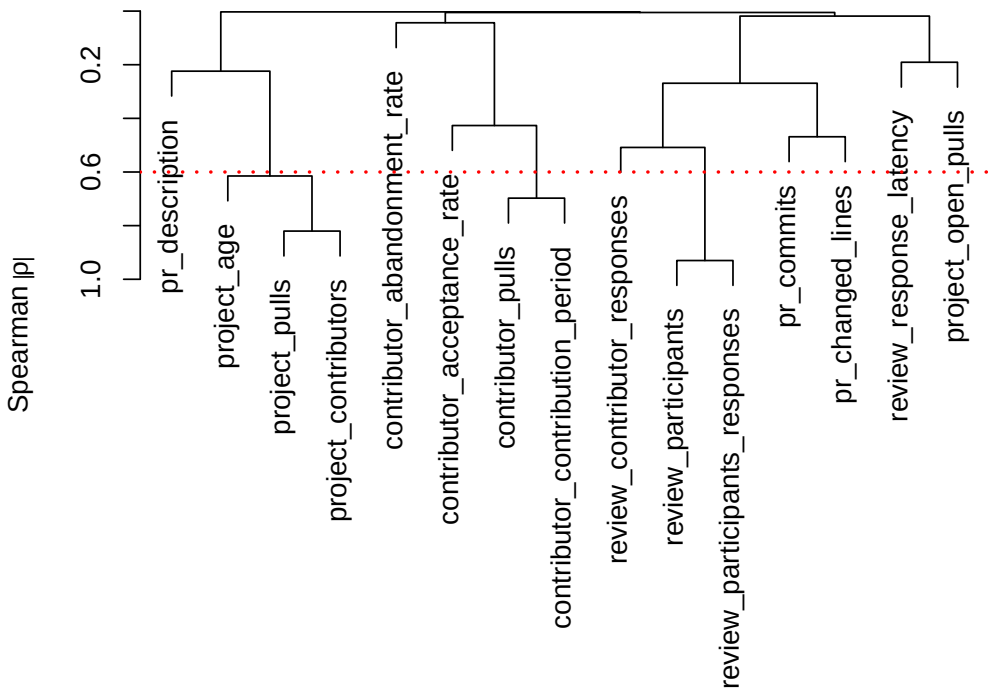


Figure 3.1: Spearman's ρ correlation among different pairs of features across the combined data of the studied projects.

relationship between each pair of features, we use Spearman's ρ [83] as a nonparametric test because we cannot assume the distribution of our features to be normal. We measure correlations on the combined data of the studied projects to ensure that any correlation exists across all of them. Figure 3.1 presents a hierarchical cluster of the correlations generated using the `Hmisc` package [84]. For each group of strongly correlated features (i.e., $|\rho| \geq 0.6$ as suggested by Evans [85]), we keep the feature that is easier to interpret for our study and remove the rest. Accordingly, we drop the following four features from our analysis: *project_pulls*, *project_contributors*, *contributor_contribution_period*, and *review_participants*.

Step 3: Remove redundant features. While we remove highly correlated features in Step 1, we also need to eliminate redundant features to focus on the most important ones. To identify redundant features, we use the `Hmisc` package [84], which applies flexible parametric additive models to measure how well each feature can be predicted from other features. Similar to our correlation analysis, we measure redundancy on the combined data of the studied projects to ensure that any redundancy exists across all of them. Accordingly, we did not find any redundant features.

Step 4: Build classifier models. To gain deeper insights on PR abandonment, we build a random forest classifier for each project using the `ranger` package [86]. To model the abandonment probability, we consider

the type of PR (i.e., abandoned or nonabandoned) as the dependent variable and the selected 11 features as the independent variables. Random forests [87] are commonly used in various domains and outperform linear models in both the predictive power and the ability to learn complex relations. To boost the predictive power of each model, we use the `tuneRanger` package [88]. This package automatically tunes the following three hyperparameters of random forests using sequential model-based optimization [89]: (i) the number of variables randomly drawn for each split, (ii) the fraction of instances randomly drawn for training each tree, and (iii) the minimum number of samples that a node must have to split.

Step 5: Evaluate performance of models. To ensure that the models are reliable for our analysis, we evaluate their predictive power using the following two recommended metrics for binary classifiers [90]:

- **AUC-ROC:** which measures the area under the Receiver Operating Characteristic (ROC) curve [91]. The ROC curve plots the true positive rate (i.e., the ratio of correctly classified abandoned PRs to truly abandoned PRs) against the false positive rate (i.e., the ratio of incorrectly classified abandoned PRs to nonabandoned PRs) across different thresholds. The value of AUC-ROC ranges from 0 to 1, with values more than 0.5 indicating better performance than a no-skill classifier (i.e., baseline). Note that the value of AUC-ROC is the same for both positive (i.e., abandoned PRs) and negative (i.e., nonabandoned PRs) classes.

- **AUC-PR:** which measures the area under the Precision-Recall (PR) curve [92]. The PR curve plots the precision (i.e., the ratio of correctly classified abandoned PRs to all classified abandoned PRs) against recall (i.e., the ratio of correctly classified abandoned PRs to truly abandoned PRs) across different thresholds. The value of AUC-PR also ranges from 0 to 1, but the performance of a no-skill classifier (i.e., baseline) is determined by the distribution of classes in a dataset (i.e., distribution of abandoned and nonabandoned PRs). Note that, unlike AUC-ROC, the value of AUC-PR is different between positive (i.e., abandoned PRs) and negative (i.e., nonabandoned PRs) classes.

To reduce bias in our performance evaluations, we perform a stratified 10-fold cross-validation with 10 repeats (a total of 100 iterations) for each model using the `mlr` package [93]. Table 3.5 presents the results of our performance evaluation for each model, where the baseline column shows the ratio of the minority class (i.e., abandoned PRs). We observe that our models have a good performance with an average AUC-ROC of 0.87 and perform at least four times better than the baseline in terms of AUC-PR.

Table 3.5: Performance scores of our model for each studied project.

Project	AUC-ROC	AUC-PR	Baseline	AUC-PR / Baseline
Ansible	0.88	0.042	0.006	7.08x
DefinitelyTyped	0.86	0.262	0.054	4.84x
Elasticsearch	0.86	0.021	0.003	6.14x
Homebrew Cask	0.96	0.063	0.001	42.78x
Kibana	0.96	0.095	0.002	50.27x
Kubernetes	0.89	0.375	0.060	6.23x
Legacy Homebrew	0.92	0.116	0.013	8.85x
Odoo	0.77	0.029	0.002	17.25x
Rust	0.81	0.109	0.020	5.50x
Swift	0.82	0.059	0.003	19.36x

Step 6: Analyze importance of features. So far, we have built classifiers that can aptly model the abandonment probability of PRs. To compare the relative importance of different features, we perform permutation feature importance analysis [94] for each model using the `iml` package [95]. This approach permutes a feature to break the association between the feature and the outcome (i.e., the abandonment probability in our case). The importance of the feature is then measured by how much error the permutated data introduces compared to the original error (i.e., loss in AUC in our case) after 100 iterations. Therefore, the most important features have the largest impact on the performance of our models and thus are more valuable for predicting PR abandonment.

Step 7: Analyze impact of features. After measuring the relative importance of the features in Step 6, we aim to describe how each feature varies the abandonment probability. For this purpose, we generate Accumulated Local Effects (ALE) plots [96] using the `iml` package [95]. ALE shows the effect of a feature at a certain value compared to the average prediction of the data. In other words, a downward trend implies a reduced probability of abandonment, an upward trend implies an increased probability of abandonment, and a stable ALE implies no changed probability of abandonment. To focus on the most common values of features, we filter out the values over the 99th percentile for each feature in each project. The plots are then created by dividing a feature into 10 intervals selected based on its quantiles. For each interval, the PRs that fall into that interval are considered for calculating the difference in their prediction when replacing the value of the feature with the upper and lower limits of the interval. We model the abandonment probability as a function of the selected features, and thus, any relationship is causal for the model and may not hold in the real world [97].

Table 3.6: Importance of different features across the studied projects.

Dimension	Feature	Ansible	DefenitelyTyped	Elasticsearch	Homebrew Cask	Kibana	Kubernetes	Legacy Homebrew	Odoo	Rust	Swift	Overall Rank	Average Loss
Pull Request	pr_changed_lines	7	8	7	**3**	6	9	**2**	7	6	9	6	1.55
	pr_description	8	**3**	**4**	**5**	8	6	9	9	7	10	8	1.48
	pr_commits	10	**5**	10	11	10	10	10	10	10	8	11	1.21
Contributor	*contributor_acceptance_rate*	**5**	**4**	**1**	6	**5**	**5**	8	**3**	**1**	**3**	**2**	1.95
	contributor_pulls	**3**	9	**3**	**4**	**3**	7	6	**4**	**4**	**1**	**3**	1.81
	contributor_abandonment_rate	11	11	11	8	11	8	11	11	9	6	10	1.24
Review Process	*review_participants_responses*	**1**	**1**	**2**	**1**	**1**	**1**	**1**	**1**	**5**	**2**	**1**	4.45
	review_contributor_responses	9	6	6	**2**	**2**	**4**	**3**	**5**	**2**	11	**4**	1.74
	review_response_latency	**2**	10	9	9	9	11	**5**	**2**	11	**5**	9	1.33
Project	*project_age*	6	**2**	**5**	7	**4**	**2**	**4**	8	**3**	7	**5**	1.73
	project_open_pulls	**4**	7	8	10	7	**3**	7	6	8	**4**	7	1.49

3.4.2 Findings

Table 3.6 summarizes the importance of different features in each project. We find that the features of the review process, contributors, and projects play a more prominent role in PR abandonment than the features of PRs themselves. Specifically, the number of responses from the participants is the most important feature by a large margin, indicating that the activity of reviewers is essential in classifying abandoned PRs. The second and third most important features are the acceptance rate and the number of previously submitted PRs of the contributor, respectively, highlighting the impact of the contributor experience on PR abandonment. The fourth and fifth most important features are the number of responses from the contributor and the age of the project. Other features, except for the abandonment rate of the contributor, are also among the top five in at least one project, showing that different features have a different impact on PR abandonment due to the inherent differences between the projects. Unexpectedly, we also observe that the number of commits in the PR, the abandonment rate of the contributor, and the latency to the first response from the participants are overall the least important features, respectively. In the following, we describe how the top five features impact the predicted abandonment probability of PRs. The ALE plots for the rest of the features can be found in Section 6.2.5.

PRs with long discussions are more likely to get abandoned. Figures 3.2 and 3.3 show how the number

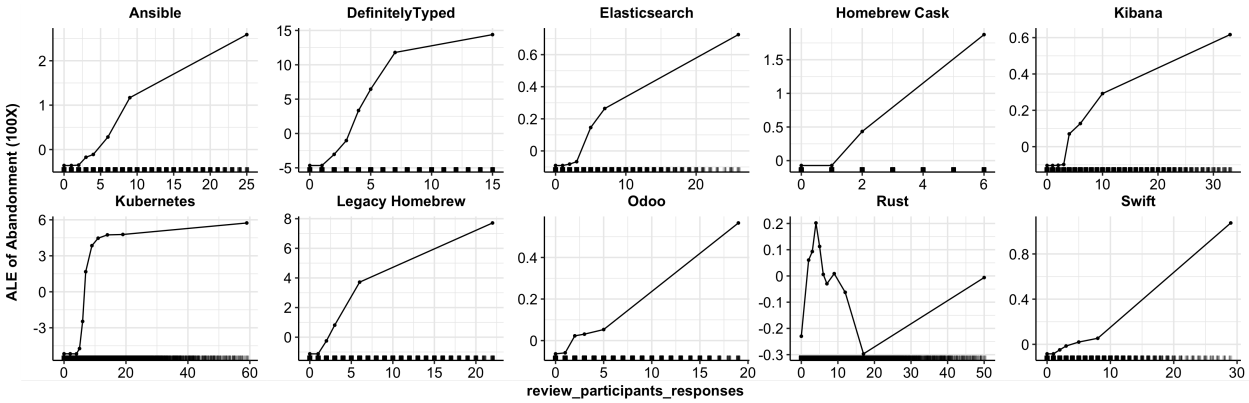


Figure 3.2: ALE plots showing how review_participants_responses varies the abandonment probability of PRs across the studied projects.

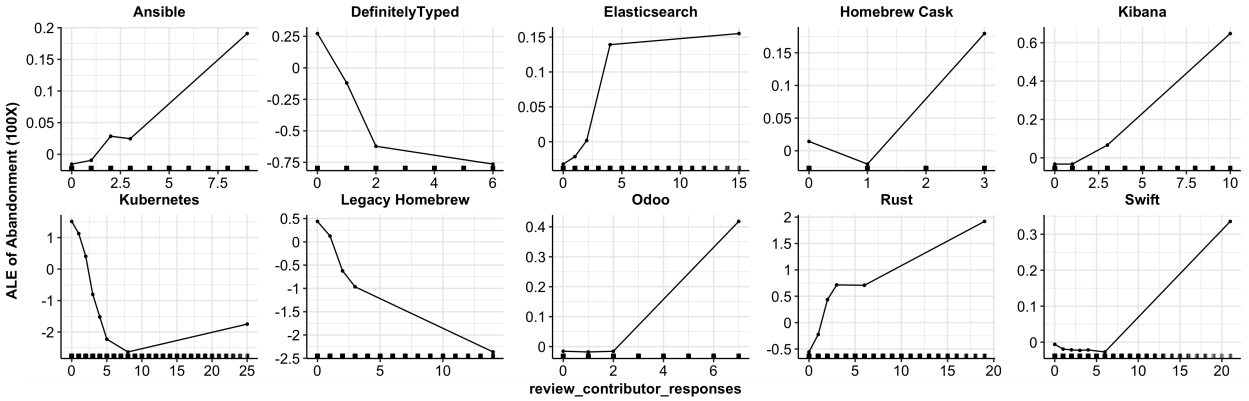


Figure 3.3: ALE plots showing how review_contributor_responses varies the abandonment probability of PRs across the studied projects.

of responses from the participants and from the contributor varies the abandonment probability of a PR across the studied projects, respectively. We find that the probability of abandonment increases in most of the projects as the number of responses from the participants or the contributor increase (i.e., upwards trend). We also observe that PRs that receive more than three responses from either the participants or the contributor have an increased probability of abandonment in most of the projects. The results provide further evidence that abandoned PRs often demand more time and effort from both their reviewers and their contributors (see Section 3.3.2).

Novice contributors are more likely to abandon their PRs. Figures 3.4 and 3.5 show how the acceptance rate and the number of previously submitted PRs by the contributor vary the abandonment probability of a PR across the studied projects, respectively. We observe that contributors with zero experience have the highest probability of abandonment in almost all the projects. Surprisingly, highly experienced contributors also

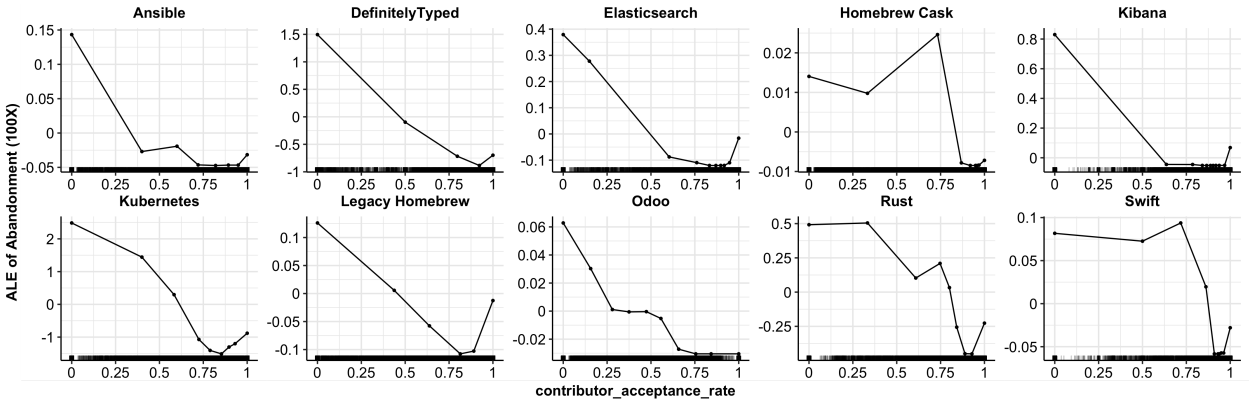


Figure 3.4: ALE plots showing how contributor_acceptance_rate varies the abandonment probability of PRs across the studied projects.

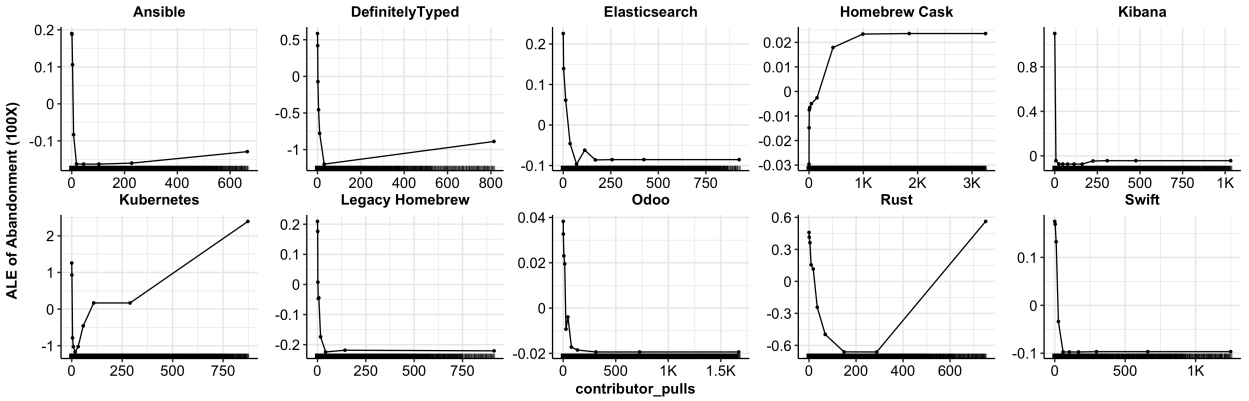


Figure 3.5: ALE plots showing how contributor_pulls varies the abandonment probability of PRs across the studied projects.

have an increased probability of abandonment in a few projects. The results suggest that the contributors of abandoned PRs may need more guidance and attention from the maintainers.

PR abandonment has changed throughout the history of projects. Figure 3.6 shows how the age of projects varies the abandonment probability of a PR across the studied projects. Similar to RQ1 (Section 3.3.2), we observe two contrasting patterns: PR abandonment improves throughout the time in some projects and worsens in some other projects. In the first group (i.e., Ansible, DefinitelyTyped, Elasticsearch, Kibana, and Odoo), the abandonment probability is highest when the project age is low (i.e., downwards trend). However, in the second group (i.e., Homebrew Cask, Kubernetes, Legacy Homebrew, and Swift), the abandonment probability increases when the project age increases (i.e., upwards trend). While this fluctuation may be associated with the expected change in the workload of projects as they grow, we found that the number of open PRs is less important to our models than the age of the project.

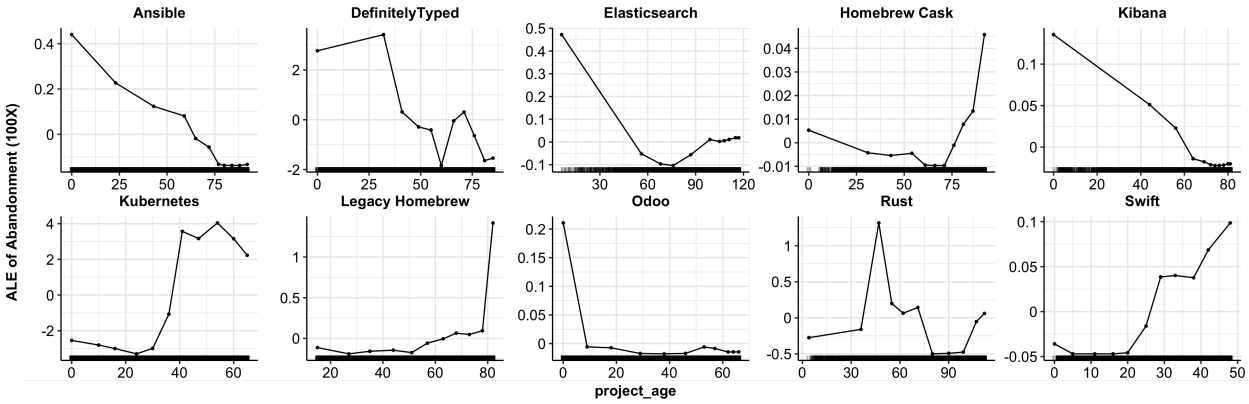


Figure 3.6: ALE plots showing how project_age varies the abandonment probability of PRs across the studied projects.

Answer to RQ2. Our findings suggest that the features of the review process, contributor, and project are more important in predicting PR abandonment than the features of PRs themselves. Specifically, PRs with more than three responses from the participants or the contributors, and those submitted by novice contributors are more likely to get abandoned. Also, the abandonment probability changes as the projects evolve, with half of the projects showing a decrease in abandonment in their mature stages and the other half showing an increase in abandonment.

3.5 RQ3: What are the probable reasons why contributors abandon their PRs in the studied projects?

In RQ1 and RQ2, we quantitatively analyzed contributor-abandoned PRs to understand how different factors influence their abandonment probability. As our last research question, we aim to complement our previous findings and gain a deeper understanding of the underlying dynamics of abandoned PRs. Specifically, we want to look for clues in the review discussions of abandoned PRs to better understand why contributors abandon their PRs.

3.5.1 Approach

We perform a manual examination to establish a taxonomy of the abandonment reasons by following the coding guidelines presented by Seaman [98]. First, we label a sample of abandoned PRs to identify the probable reasons why contributors abandon their PRs and then calculate our interrater agreement. In the

following, we explain each step in more detail:

Step 1: Identify abandonment reasons. First, we randomly select 354 PRs from 4,450 abandoned PRs of the studied projects (confidence level of 95% with a ±5% confidence interval). Then, the first three authors are required to manually examine the discussion comments of each PR and try to pinpoint the primary reason(s) why its contributor has abandoned the PR. In most cases, the contributor has abandoned the PR without any explanation, and thus we look for clues in the interactions between the contributor and the reviewers to identify the most probable reasons for the abandonment. In cases where the contributor provided a reason for their abandonment decision, we also investigate other major reasons that might have led to their abandonment.

We perform the labeling in two rounds. In the first round, three annotators independently label a random sample of 60 PRs from the selected 354 PRs to establish the classification scheme. In the second round, we divide the sample 354 abandoned PRs (including the 60 PRs from the first round) into two sets to be labeled using the classification scheme from the previous round: the first set was labeled independently by the first and second authors, and the second set was labeled independently by the first and third authors. Finally, the annotators merged the labels and further refined the labels. In each round, when the annotators had different opinions, they discussed until they reached an agreement and then retroactively updated all the previously labeled PRs to ensure a coherent classification.

Step 2: Calculate interrater agreement. To ensure the quality of our taxonomy, we calculate the Cohen's Kappa coefficient [99] using the `scikit-learn` package [100]. This statistic is commonly used to evaluate the interrater agreement in different domains. The value of Cohen's Kappa ranges from -1.0 to $+1.0$, with values more than 0 indicating an agreement better than chance. We obtained a Kappa score of 0.73, which is considered a substantial agreement as suggested by Landis and Koch [101].

3.5.2 Findings

As shown in Table 3.7, we identified 10 major reasons why contributors abandon their PRs, grouped into three categories: (i) contributor-related reasons, (ii) maintainer-related reasons, and (iii) PR-related reasons. Note that the total frequency of the identified reasons is greater than 100% because we observe multiple reasons for some abandoned PRs. We find that the most frequent abandonment reasons are related to the obstacles faced by contributors followed by the hurdles imposed by maintainers during the review process. Particularly, difficulty addressing the maintainers' comments, lack of review from the maintainers,

Table 3.7: Probable reasons why contributors abandon their PRs.

Category	Reason	Frequency (%)
Contributor-related	Difficulty addressing the maintainers' comments	45.8
	Difficulty resolving the CI failures	20.9
	Difficulty resolving the merge issues	14.1
	Difficulty complying with the project requirements	1.4
Maintainer-related	Lack of review from the maintainers	22.6
	Lack of answer from the maintainers	9.3
	Lack of integration by the maintainers	6.5
	Lack of consensus among the maintainers	4.5
PR-related	Existence of duplicated work	3.1
	Dependency on upcoming changes	1.4

and difficulty resolving the CI failures are the most frequent reasons (observed in more than 20% of abandoned PRs). In the following, we discuss the identified reasons in the order of their frequency.

Difficulty addressing the maintainers' comments (45.8%). In almost half of the abandoned PRs, their contributors found it difficult to address the maintainers' comments, questions, or change requests. The contributors did not have the required technical knowledge or enough time to continue the work (e.g., [P198] said *"Nope. Had no time and will to take on this."*). Interestingly, contributors may ask others to continue the work (e.g., [P340] said *"If you don't mind taking it over, that would be fantastic. Happy to provide whatever help I can!"*).

Lack of review from the maintainers (22.6%). The second common reason why contributors abandon their PRs is that they did not receive a timely (if any) review from the maintainers. Sometimes, a PR gets reviewed, and the contributor addresses the maintainers' comments, but the maintainers do not follow it up. For example, [P7] was closed after multiple rounds of discussions, even though the contributor addressed the maintainers' comments (*"Oh dang, I guess it doesn't warn you after the first time. I'll get this re-made soon and try to be a little more proactive about getting input."*). Not receiving timely reviews from the maintainers may also send a signal to the contributors that their work is not treated seriously (e.g., [P182] said *"I'm happy to resolve these merge conflicts now, but before I do, I am curious if I have messed something else up in my contribution? Since this didn't get a comment since July of last year, I am afraid I've missed something crucial."*). If PRs are not reviewed in a timely manner, they may become outdated and require extra work from their contributors to rebase and update them.

Difficulty resolving the CI failures (20.9%). The third reason includes cases where the contributors

found it difficult to resolve the Continuous Integration (CI) failures arisen during the review process. Such failures are often brought up by the project bots even before the PR gets a review from the maintainers. Sometimes, the contributors do not even know how to fix the CI failures and ask the maintainers for help (e.g., [P346] said *"I've looked at the errors in Travis. Most of them seem unrelated to the PR. Could you please help me out with this?"*).

Difficulty resolving the merge issues (14.1%). The fourth reason includes cases where the contributors found it difficult to resolve the merge issues arisen during the review process. If the project codebase has been updated, the contributors are asked to rebase their local branch, resolve any merge conflicts, and then push their changes again. Such issues typically arise when the maintainers take a long time to review the PR, and the PR becomes outdated (e.g., [P181] said *"I am willing to keep rebasing (and certainly willing to continue responding to comments), but not without some indication that it will eventually be merged."*). We also observe that contributors are sometimes asked to squash their pushed commits into a single commit to reduce the noise in the revision history, which also requires additional effort and time from them.

Lack of answer from the maintainers (9.3%). In some cases, we observe the reason for abandonment is because the contributors had not received a timely (if any) answer from the maintainers when they asked for help to complete a task (e.g., [P300] said *"CI is timing out. Everything is fine until the timeout. Anything I can do to get this merged?"*) or asked for clarification (e.g., [P313] said *"Modifying the passed in proxyTransport cannot be considered 'safe'?"*) or asked for confirmation (e.g., [P273] said *"Is that the right way?"*). Note that this reason is different from "lack of review from the maintainers" as such PRs are blocked because the contributor is awaiting an answer to a question from the maintainer and not awaiting a review for the applied changes.

Lack of integration by the maintainers (6.5%). This reason includes cases where a PR has been already approved by the reviewers but has been pending integration. In some projects, such as Kubernetes, a PR should undergo a two-phase review process. In the first round, reviewers approve the changes, and then project integrators need to merge the changes. However, contributors may abandon their PRs if the integrators do not attempt to merge the PR in a timely (if any) manner after the reviewers have approved the PR. For example, in [P1], the PR has been approved for integration multiple times. However, since the integrators were not responsive, the PR needs to be rebased, requiring additional work from the contributor. In [P307], a reviewer suggested *"Might be worth pinging the reviewers too if that is all that is stopping progress."*

Lack of consensus among the maintainers (4.5%). This reason includes cases where the reviewers

could not reach a consensus on how to continue with the PR. This disagreement typically arises when there is no straightforward solution to resolve the PR issues, and each alternative has its own advantages and disadvantages. Such PRs often undergo a long discussion and demand lots of time and effort from both reviewers and contributors (e.g., [P354]). However, the PR eventually gets abandoned by the contributor due to inconsistent feedback and often overlong discussions.

Existence of duplicated work (3.1%). This reason includes cases where the work is either a duplicate of an existing PR (e.g., [P294] said *"I am abandoning this PR in favor of npm-ramda, but I wont close it in case someone want to make use of it."*), another contributor has submitted a more comprehensive PR, or the issue addressed in the PR is no longer applicable (e.g., [P340] said *"pkg-config should no longer be able to pick up non-deps under superenv. Hopefully that means this is resolved."*).

Difficulty complying with the project requirements (1.4%). This reason includes rare cases where the contributors found it difficult to comply with the project-specific requirements. In projects such as Kubernetes, the contributors are asked to sign the contribution level agreements before their PR even gets reviewed by the maintainers (e.g., [P149]). Also, many projects provide templates for the PR description and ask the contributors to update the description according to the templates (e.g., [P134]).

Dependency on upcoming changes (1.4%). In rare cases, the abandoned PRs were not valuable on their own and depended on other changes that must be merged first before the proposed changes can be considered (e.g., [P11] said *"Hopefully, once mappable and partial types land in TS, I will fix this."*).

Answer to RQ3. Our findings suggest that the most frequent abandonment reasons are related to the obstacles faced by contributors followed by the hurdles imposed by maintainers during the review process. Specifically, difficulty addressing the maintainers' comments, lack of review from the maintainers, difficulty resolving the CI failures, and difficulty resolving the merge issues are the most common reasons why contributors abandon their PRs.

3.6 Perspectives of the Project Maintainers

To gain deeper insights on PR abandonment, we design a survey asking the core maintainers of the studied projects about their perspectives on our findings and how to tackle PR abandonment. After explaining the goal of this research and presenting a summary of our findings, we ask the following three open-ended questions in

the survey:

- Does your team implement any approaches to deal with abandoned PRs? If so, what approaches does your team use?
- Do you suggest any approaches that can minimize the risk of a PR being abandoned by its contributor? (these can be approaches that your team does or does not use).
- Do you have any feedback about the four findings of our study? (we would love to hear it, positive or negative).

We send emails to the top 25 core developers of each studied project (a total of 250 emails) to invite them to participate in our survey. From these invitations, we receive a total of 16 responses (6.5%) to our survey. Our response rate is similar to the 5% response rate, commonly observed in software engineering studies [102]. From the 10 studied projects, we received responses from all the projects except Elasticsearch. As our sample is small, we refrain from discussing particular projects and instead present the overarching themes that appeared in the participant responses.

3.6.1 How do projects *deal* with abandoned PRs?

In this question, we aim to understand the processes and practices that the projects have already put in place to deal with abandoned PRs. Out of the 16 participants, six responded that their team had implemented approaches to deal with abandoned PRs. These approaches are:

Holding triage meetings (4x). The most commonly mentioned approach to deal with abandoned PRs is holding (recurrent) triage meetings, where developers review the status of PRs and support PRs that need attention. According to the survey participants, triage meetings streamline the communication and help find problems before it causes the PRs to become abandoned.

> *"We have regular triage meetings to review the status of PRs that abandoned or about to be abandon. We also help new comers and new maintainers for reviving the old PRs and merge them."* [S1]

> *"Better upfront planning and communication help eliminate abandoned PRs. Many times, a quick zoom with a teammate to talk through the proposed change goes a long way into finding problems before a large effort is needed."* [S6]

Using bots to auto-close abandoned PRs (2x). While all the projects use Stale bot [11] or a similar in-house implementation (e.g., fejta-bot in Kubernetes [103]) to follow up with PRs that are about to get abandoned and auto-close already abandoned PRs, two participants explicitly mentioned their use of such bots. As a respondent explained:

> *"A bot first pings owners of the code after a week. Then it pings the submitter a couple of weeks later, telling them that it will be closed soon."* [S2]

Most of the participants (10 responses) reported that their project does not implement any particular approach to deal with abandoned PRs. Of these participants, four explained why their team had not adopted any specific approach to prevent PR abandonment:

The maintainers are overwhelmed with work (2x). Two maintainers reported that they are overwhelmed with work and thus have decided to reduce their efforts on retaining PRs from external contributors. As a respondent explained:

> *"We are overwhelmed with work, and most community PRs have a low ration value/effort needed to merge it, so we essentially gave up, except for rare cases."* [S13]

The project does not rely on open source contributions (2x). Two participants mentioned that their project does not rely on external contributions for the implementation of new features or improvements. Therefore, there is little incentive to spend special time and effort in retaining PRs from external contributors. As a respondent explained:

> *"We appreciate community PRs, but the vast majority of the contributors to our project are employees and so we don't rely on open source contributions for features or improvements. Community PRs are usually applicable to a specific niche usecase which we'd be happy to accept if the contributor is willing to go through the process with us."* [S5]

3.6.2 How do maintainers *recommend* to mitigate PR abandonment?

In this question, we aim to understand the approaches that the maintainers recommend to minimize the risk of PRs getting abandoned by their contributors, whether their team has adopted them yet or not. Out of the 16 participants, 15 responded with suggestions that they believe could mitigate PR abandonment. In

the following, we summarize these recommendations into suggestions for contributors and suggestions for maintainers.

Maintainers should strive to make the contributor experience as smooth as possible (6x). It stands to reason that the more obstacles contributors face, the higher the chances of abandonment. One participant (S5) mentioned that having helpful and understandable error messages can help contributors fix issues in their PRs. Another participant (S3) mentioned that maintainers should "put kid gloves on" when dealing with newcomers and make community contributions as painless as possible. One participant (S15) even suggested that maintainers merge PRs with minor issues and then either make the required changes themselves or open an issue in the project. As one respondent aptly summarizes:

> *"Responding quickly and encouragingly, and making your PR contributor experience as seamless as possible (automatic CI, helpful and understandable error messages) are likely all you can do. It's a lot of work even to submit a PR, the extra work necessary when updates are needed isn't something most people will be willing to give."* [S5]

The participants also mentioned that improving the project's testing documentation (S1) and contribution guidelines (S2), and making bot instructions more understandable (S2) could help to mitigate PR abandonment. One participant (S13) also cited that projects need to increase their available resources, with another participant (S1) mentioning that increasing community reach and having maintainers from the community may help maintainers better handle the required workload.

Maintainers should establish a triage process for external contributions (2x). Once again, the participants mentioned the importance of a triage process in mitigating PR abandonment. A triage process helps assign reviewers to a PR based on their expertise and experience and may lead to timely responses to contributors. One participant (S14) suggested that PRs should preferably be assigned to one reviewer instead of an entire team to compel reviewers to act and prevent idleness. Another respondent described that the triage process should also monitor the status of PRs and act if reviewers have not responded to the contributors yet:

> *"[Projects should have] a human rotation that triages pull requests and checks if they are progressing, or if someone has dropped the ball or is waiting on some event that will never happen."* [S16]

Contributors should create PRs that are clear and concise (3x). The participants emphasized the importance of PRs to focus only on a single use case and provide a clear description of changes. PRs that

include multiple unrelated changes create a burden for reviewers that need to ensure all changes are correct. As two participants responded:

> *"Make the proposal clear and concise. The reviewer might take 10 or 15 seconds to figure out the problem being solved and the solution. If it takes longer, the reviewer will probably give up."* [S11]

> *"Smaller PRs are obviously better. We try not to nit-pick though it is human nature that a 100 line diff gets no nitpicking and a 4 line diff gets plenty. This can hardly make people want to contribute and is unfortunate."* [S9]

Contributors should assess the project's interest in the proposed changes before submitting PRs (3x). Managing expectations is important in contributing to open source projects. As different projects have different philosophies and needs regarding community contributions, contributors should assess whether maintainers value their PRs. Typically, new features are harder to integrate than bug fixes and performance improvement patches.

> *"In large projects, a performance improvement or a bug fix proposed by an external contributor is more likely to be merged than a new feature. Not only because it's usually simpler, but because the feature might not be in line with the project's interests. A bug fix or a performance improvement is always in line with the project's interest."* [S10]

> *"Managing expectations could be a factor. We won't merge entirely new features just like that, simply because it is really important that Odoo remains and improves on being kept simple. E.g. PRs could come from a client project that needs a certain feature, but merging it like this, might do harm to a lot of other clients or cause more further problems where we need some distance to really think about the best solution."* [S12]

One way to assess the validity of the contribution is to open an issue in the project. The contribution guidelines of the majority of the projects explicitly state that contributors should first open an issue to discuss their contributions and defer the implementation until when the maintainers have agreed on the usefulness of the proposed changes. As a participant responded:

> *"Opening issues to discuss changes prior to posting PRs helps reduce abandoned PRs. Discussing the change ahead of time gives developers and the community time to explore the change and ensure that the 1) proposed change is one that the project wants to maintain, 2) proposed change is scalable, 3) proposed change is a feature that is needed by many use cases and not a one off for specific use case, 4) proposed implementation is maintainable and fits with the architecture and future of the project."* [S6]

Both contributors and maintainers should be more upfront about their intentions (2x). Two participants stated that communication should be improved from both sides. Maintainers need to be upfront about their intentions on merging (or not) the contribution from contributors to avoid wasting effort and time. As a participant responded:

> *"There needs to be a clear signal from the project to the PR contributor if there is no interest at all in the PR, or if there is interest, what needs to be fixed to be accepted. Then participate or help, or signal when the effort can no longer be sustained. If there is no clear signal, the contributor has no idea what's going on. Automated "closer-bots" cannot solve this problem (or make it worse)."* [S8]

Similarly, contributors should mention their willingness to make the requested changes. One participant mentioned that anxiety could play an important factor leading to PR abandonment, particularly when miscommunication happens to newcomers to the project:

> *"Anxiety about contribution probably leads to some abandonment. I suggest to all newish contributors that they remember the people in charge of these projects where just like them once."* [S9]

3.6.3 How do developers *interpret* our findings?

We provided participants with a pre-print of this manuscript and encouraged participants to report any negative or positive remarks they had about our findings. Out of the 16 participants, 14 participants commented on our findings (87.5%). Table 3.8 overviews the explanations provided by survey participants regarding each of our main findings. In the following, we discuss the survey responses for each of our findings in more detail.

Table 3.8: Overview of the explanation of our findings based on our survey responses.

Study Findings	**Survey Explanations**
Complex PRs are more likely to get abandoned.	• Complex PRs are less likely to get a timely review. • Contributor becomes frustrated by frequent change requests.
Novice contributors are more likely to abandon their PRs.	• Maintainers expect quality changes regardless of the contributor experience. • Novice contributors may find the contribution process difficult.
PRs with long discussions are more likely to get abandoned.	• Long discussions often indicate a controversial PR. • Lack of unanimous decisions lengthens the review process.
Projects have a significant influence on the likelihood of PR abandonment.	• Maintainer team structure, attitude, and workload influence PRs. • Project scope, architecture, and ownership influence PRs.

Complex PRs are more likely to get abandoned. Our findings suggest that PRs with lengthier descriptions, more commits, or more changed lines are more likely to get abandoned (RQ1–RQ2). The participants argued that such complex PRs require extra efforts from both their contributors and reviewers. Therefore, such PRs might linger for a while before getting reviewed, and also might require more changes from the contributor to become satisfactory:

> *"The more complex a PR is, the less likely a reviewer is going to spend valuable time on it, in particular if the contributor is not well known."* [S11]

> *"Either the maintainers leave them open for months or the contributor is frustrated by the numerous requested changes."* [S9]

Novice contributors are more likely to abandon their PRs. Our findings suggest that contributors who submitted fewer PRs, have a lower acceptance rate, have a lower contribution period, or have a higher abandonment rate within a project are more likely to abandon their PRs (RQ1–RQ2). The participants suggested that this can be due to projects expecting high-quality changes (with proper formatting, documentation, and description of changes) that often require access to experienced maintainers. However, projects can lower their expectations from external contributors:

> *"Like any wall in life, it is scary and something to throw yourself against. The project can help-making contributions feel more welcome. Unfortunately if you are too welcoming to contribution you get rapidly overwhelmed by contribution and cannot accept it all."* [S9]

> *"The main issue I see with abandoned PRs is that the bar for a commit is too high. To give you some perspective, new developers in my team take a few weeks to land their first commit. And that is with constant access to experienced developers/mentors. This is because we expect our commit to have: proper tests, linting, inline documentation, references to relevant commits, documentation in some cases, good commit message, and targetting the correct branch. I think we should lower the bar for external contributors (so they can quickly land a fix), but eventually, still add an additional commit with a test or something if needed. Clearly, doing that requires some resources. I tried doing that a few years ago, and was quickly slammed with pings everywhere to ask me to work on those PRs!"* [S13]

PRs with long discussions are more likely to get abandoned. Our findings suggest that PRs involving more participants, more responses from the participants or from the contributor, or with higher latency to get

a response from a reviewer in a project are more likely to get abandoned (RQ1–RQ2). The participants argued that long discussions could indicate a controversial PR lacking a unanimous solution to address the PR issues:

> *"Long discussions themselves are not a problem. However, they are often indicative of disagreement, a complex topic, or an absence of a solution (the solution in the PR is not correct, but commenters on the PR don't have a good suggestion either)."* [S16]

> *"[long discussions] seems like a proxy for controversial PRs, which I suspect is also a factor at play. In my personal experience, open source projects are often lead by passionate individuals with strong opinions about the way things should be done, which often conflict with the opinions of new people to the project. I think this could be addressed by discussing changes beforehand but this is another hurdle which makes contributing more challenging."* [S5]

However, a strong leadership team could prevent such controversial PRs from becoming extra lengthy:

> *"Good projects have strong leaders that make decisions and don't deliberate too long. Bitcoin suffered here greatly after Satoshi left."* [S9]

Projects have a significant influence on the likelihood of PR abandonment. Our findings suggest that projects have a significant influence over PR abandonment and that throughout the history of projects, the rate of PR abandonment has significantly fluctuated as the projects evolved. The participants suggested that such fluctuations might be related to changes in the attitude of the team, scope and architecture of the software, and popularity and ownership of the project:

> *"It's a multitude of factors. Attitudes of maintainers. Software architecture of the projects (if well designed you can expect well designed PRs). Scope of the projects (ie. documentation is clear on the scope to prevent PRs that feature creep). Fame: too big a project will get too much contribution and likely there will be insufficient people to manage it."* [S9]

> *"You might also take into account the ownership of the project: is it a community project? A company project? Who is writing the roadmap of the project? A company project has a roadmap in line with its business development, which is not the case for a community project." [S10]*

3.7 Discussion

Combining the results from our quantitative (RQ1–RQ2) and qualitative (RQ3) investigation provides evidence that contributors and the review process play a more prominent role in PR abandonment than projects and PRs themselves. In the following, we integrate the findings from our three research questions and our survey with core developers and further discuss the implications of our findings.

3.7.1 The Role of Contributors in PR Abandonment

Our findings indicate that the contributors of abandoned PRs usually have less experience than the contributors of nonabandoned PRs. Specifically, we observed that novice contributors who have submitted fewer PRs, have a lower acceptance rate, or have a lower contribution period within a project are more likely to abandon their PRs in most of our studied projects (RQ1–RQ2). Our survey results suggest that novice contributors often find the contribution process more difficult as maintainers typically expect high-quality changes (with proper tests, formatting, documentation, and description of changes) regardless of contributor experience before approving the changes to get merged (Section 3.6). We also observed that the contributors of abandoned PRs have frequently faced many obstacles (due to lack of enough knowledge, time, or even interest) to continue and complete the review process (RQ3). Indeed, inexperienced contributors face various barriers in making their contributions accepted [104, 105, 106]. Prior studies have also reported the positive impact of contributor experience in acceptance and review time of PRs [1, 37, 38, 39]. Our survey respondents recommend maintainers either lower their expectations or be more attentive and supportive towards external contributors (especially casual contributors or newcomers) throughout the review process. Also, contributors can discuss their proposed changes before submitting a PR to facilitate the review process, especially if the change introduces new features or involves large changes. This discussion helps contributors to ensure that their proposed changes align with the project roadmap and design. Contributors are also expected to adhere to contribution guidelines and project conventions as it helps them have a better grasp of the review process.

3.7.2 The Role of Review Processes in PR Abandonment

Our findings indicate that the review process of abandoned PRs is usually lengthier than the review process of nonabandoned PRs. Specifically, we observed that lengthy PRs, which involve more participants, or more responses from the participants or from the contributor are more likely to get abandoned in most

of our studied projects (RQ1–RQ2). Our survey results suggest that long discussions are often indicative of a controversial PR that is addressing a complex issue or does not have a unanimously accepted solution (Section 3.6). We also observed that abandoned PRs frequently lack a (timely) review, response, or even action from the maintainers, which also unnecessarily lengthens the review process of a PR (RQ3). Prior studies have also reported that high response latencies and lengthy discussions negatively impact the acceptance and review time of PRs [36, 38, 39]. Our survey respondents recommend maintainers be more responsive and support external contributors (especially casual contributors or newcomers) till the completion of their PRs. In cases that a PR needs only trivial changes, maintainers can merge the PR as is and either implement the changes themselves or open a new issue for the required changes. Also, maintainers can hold recurrent triage meetings to review the status of PRs and support PRs that need attention to mitigate PR abandonment. In cases where a PR has become lengthy, lead maintainers should involve and decide on the outcome of the PR using a voting process.

3.7.3 The Role of Projects in PR Abandonment

Our findings indicate that projects have a significant influence over PR abandonment. Specifically, we observed that the rate of abandoned PRs has significantly fluctuated throughout the history of projects, with some projects constantly decreasing the abandonment rate as they become mature, i.e., Ansible, Kibana, and Odoo (RQ1–RQ2). Our survey results suggest that projects typically undergo changes in their team, size, architecture, scope, policies, practices, or even ownership during their development lifecycle. Such changes bring with them both positive and negative aspects, which can fluctuate the rate of PR abandonment (Section 3.6). Prior studies have also reported that project maturity has a mixed impact on the acceptance and review time of PRs [36, 38]. Our survey respondents recommend projects streamline their contribution process as much as possible to better accommodate new contributors.

3.7.4 The Role of PRs in PR Abandonment

Our findings indicate that abandoned PRs are usually more complex than nonabandoned PRs. Specifically, we observed that complex PRs, which have lengthier descriptions or more commits are more likely to get abandoned in most of our studied projects (RQ1–RQ2). A PR can be complex at submission time, when it contains too many commits or an abnormally lengthy description, or become more complex as its contributor

submit additional commits (and thus makes more changed lines) during the review process to address the changes requested by the maintainers. Our survey results suggest that a complex PR is more likely to linger for a while before getting a first or even a follow-up review, especially if its contributor is not well-known to the maintainers (Section 3.6). We also observed the lack of review from the maintainers as a frequent reason among abandoned PRs (RQ3). Prior studies have also reported that complex PRs negatively impact their acceptance and review time [36, 37, 38, 39]. Our survey respondents recommend contributors make their PRs clear, concise, and focused as complex PRs are more difficult to review and require more interactions with the contributor to become ready. Also, maintainers expect PRs to have proper tests, formatting, documentation, and description of changes according to the project requirements.

3.8 Limitations

3.8.1 Threats to Internal Validity

Threats to internal validity are concerned about the issues that might affect the validity of our findings. The first threat is related to our definition of abandoned PRs. We define abandoned PRs as those promising PRs that have been neither integrated nor rejected because their contributors have left the review process unfinished. While in our preliminary investigation, we rarely found cases where another developer continues an abandoned PR, but this can be systematically investigated in future studies. The second threat is related to the process of identifying abandoned PRs. Our heuristics may have missed some truly abandoned PRs and wrongly marked some PRs as abandoned. To mitigate this threat, we considered as many relevant keywords as possible by iteratively refining our keywords as we observed new patterns in the discussion comments of known abandoned PRs. Also, we assessed the quality of our dataset by manually investigating 100 abandoned PRs. The third threat is related to the process of identifying the reasons why PRs get abandoned by their contributors. We may have drawn wrong conclusions in card sorting because the coders may have had preconceptions. To minimize this bias, each PR was independently labeled by at least two authors, and then the three authors discussed and merged the labels. The fourth threat is related to the completeness of the abandonment reasons. To further minimize this risk, we coded all the remaining cards when saturation was reached in card sorting. We also performed a second pass over all cards to ensure that we did not miss any important information.

3.8.2 Threats to External Validity

Threats to external validity are concerned with the generalizability of our findings across different projects. To conduct our study, we focused on 10 popular GitHub projects with the richest historical PR data. Although the studied projects cover several different application domains and programming languages, they do not represent the entire open-source ecosystem. Therefore, our findings may not generalize beyond our studied projects, especially since we observe conflicting patterns across different projects due to their inherent differences. Future replication studies with a more diverse selection of projects both inside and outside the open-source ecosystem are required to obtain more widely applicable insights. Also, our survey findings are based on the responses from 16 participants. While these participants are all among the top core maintainers of the studied projects, different maintainers may have different perspectives, and thus our findings may not be generalized to other settings.

3.9 Chapter Summary

Abandoned PRs waste the time and effort of their contributors and their reviewers. To provide more comprehensive insights into the underlying dynamics of PR abandonment, we conducted a mixed-methods study on 10 popular and mature GitHub projects. Using statistical techniques, we found that abandoned PRs tend to be more complex, their contributors tend to be less experienced, and their review processes tend to be lengthier than nonabandoned PRs. We then relied on machine learning techniques to determine the relative importance of the features and describe how each feature varies the predicted abandonment probability of PRs. We found that the features of review processes, contributors, and projects are more important for predicting PR abandonment than the features of PRs themselves. Specifically, PRs with more than three responses from either the participants or the contributors, and those submitted by novice contributors are more likely to get abandoned. Also, the abandonment probability changes as projects evolve, with half the projects showing a decrease in abandonment in their mature stages and the other half showing an increase in abandonment. To identify the probable reasons why contributors abandon their PRs, we manually examined a random sample of abandoned PRs. We found that difficulty addressing the maintainers' comments, lack of review from the maintainers, difficulty resolving the CI failures, and difficulty resolving the merge issues are the most common reasons why contributors abandon their PRs. Finally, we surveyed the top core maintainers of the studied projects to gain additional insights on how they deal with or suggest dealing with abandoned PRs and their

perspectives on our findings. Combining the findings from our research questions and survey responses, we discussed the role of PRs, contributors, review processes, and projects in PR abandonment.

Nevertheless, as mentioned by our surveyed maintainers, Stale bot is commonly used to automatically deal with abandoned PRs. Therefore, in the next chapter, we aim to better understand if and how Stale bot helps the pull-based development workflow of large open-source projects.

Chapter 4

Understanding the Helpfulness of Stale Bot for Pull-based Development

4.1 Introduction

Open-source projects widely adopt pull-based development as a more efficient and effective alternative to traditional methods for contributing and reviewing code changes [1, 2]. In this development model, contributors submit a PR to suggest changes for integration into the project. The PR is then reviewed by the project maintainers and updated by the contributor until it is ready to be merged. However, the review process of some PRs is left unfinished due to either the contributor not addressing the maintainers' comments or the maintainers not following up on the progress of the PR [24, 8, 107]. If neither progressed nor resolved, such inactive PRs accumulate over time, clutter the list of PRs, and eventually make it difficult for the maintainers to manage and prioritize unresolved PRs [8, 9]. As a real-world example, a large backlog of unresolved PRs led the DefinitelyTyped project to declare "bankruptcy" in June 2016. Consequently, they closed all unresolved PRs submitted before May 2016 just to be able to start afresh [10].

Manually keeping track of inactive PRs, following up on their progress, and closing them if needed places an additional burden on the project maintainers who are already occupied with other development tasks [24, 8, 9]. To free the maintainers from manually triaging such PRs, Stale bot [11] was released in 2017 and since then has been increasingly adopted by open-source projects on GitHub[1]. As shown in Figure 4.1, Stale bot aims to make the status of open and not progressing PRs explicit by automatically labeling, commenting,

[1]We observed that on average 7% more projects had adopted Stale bot each month till October 2021.

Figure 4.1: An example prompt by Stale bot.

and closing PRs after a pre-configured period of inactivity [13]. Nevertheless, there are ongoing debates within the open-source community on whether using Stale bot alleviates or exacerbates the problem of inactive PRs. The creators of Stale bot claim that based on the experience of hundreds of projects and organizations, Stale bot is an effective method for focusing on the work that matters most [13]. Conversely, part of the community regards Stale bot as "harmful" [14] and a "false economy" [15]. They argue that while Stale bot may initially seem helpful, it results in duplicated PRs, fragmented information, and eventually frustration in the community. Some studies have also incidentally mentioned that Stale bot can introduce noise and friction for both the contributors and the maintainers [16, 17, 18, 19, 20].

Despite all these positive and negative claims, the helpfulness of Stale bot has not yet been empirically validated. Therefore, we set out to better understand if and how adopting Stale bot helps open-source projects in their pull-based development workflow. This investigation is particularly important as Stale bot is commonly used to deal with inactive or abandoned PRs. Towards this goal, we perform an empirical study [108, 58] of 20 large and popular open-source projects on GitHub that have used Stale bot for at least one consecutive year. Specifically, we aim to answer the following research questions in this chapter:

RQ1: How much do the studied projects use Stale bot to deal with their PR backlog? We analyze the configuration and activity of Stale bot to understand the extent to which large open-source projects rely on Stale bot to automatically deal with their unresolved PRs. Our results show that the usage level of Stale bot widely varies among the studied projects. On average each month, Stale bot intervened in less than 25% of open PRs in nine projects, between 25% and 50% of open PRs in five projects, and more than 50% of open PRs in six projects. Unexpectedly, the projects with a larger backlog of unresolved PRs have typically not relied more aggressively on Stale bot to deal with their unresolved PRs.

RQ2: What is the impact of Stale bot on pull-based development in the studied projects? We apply interrupted time-series analysis [27] as a well-established quasi-experiment to understand if and how adopting Stale bot improves the efficiency and effectiveness of the pull-based development workflow in large open-source projects. Our results show that the studied projects closed more PRs within the first few months of adopting Stale bot, but overall closed and merged fewer PRs afterward. The adoption of Stale bot is also associated with faster first reviews in merged PRs, faster resolutions in closed PRs, slightly fewer updates in merged PRs, and considerably fewer active contributors in the projects.

RQ3: What kind of PRs are usually intervened by Stale bot in the studied projects? We analyze the characteristics of PRs intervened by Stale bot, as well as their contributors and review processes, to understand the factors that are associated with a higher probability of getting intervened by Stale bot in large open-source projects. Our results show that Stale bot tends to intervene more in complex PRs, PRs from novice contributors, and PRs with lengthy review processes. Specifically, besides the resolution time of PRs, the largest differences are observed in the number of prior PRs by contributors, the mean response latency of PRs, the acceptance rate of contributors, and the contribution period of contributors.

Our findings imply that adopting Stale bot helped the studied projects deal with their accumulated backlog of unresolved PRs. Stale bot has also improved the efficiency of the review process of PRs by helping the maintainers focus on PRs that are more likely to get merged. Despite these advantages, the adoption of Stale bot also brings some disadvantages. For example, the projects experienced a decrease in their number of active contributors after the adoption. Besides, Stale bot also tends to intervene more in PRs submitted by novice contributors. However, such contributors are the ones who face the most barriers and thus need the most guidance from the maintainers [109, 105, 104]. Prior studies have also highlighted the importance of attracting newcomers to ensure the sustainability of projects [110]. Therefore, relying solely on Stale bot to deal with inactive PRs may lead to decreased community engagement and an increased probability of contributor abandonment. In conclusion, our study provides a better understanding of the potential benefits and drawbacks of employing Stale bot within a pull-based development workflow.

4.1.1 Our Contributions

In summary, we make the following contributions in this chapter:

- To the best of our knowledge, this is the first in-depth study investigating the helpfulness of Stale bot for the pull-based development workflow in large open-source projects.
- We provide empirical evidence on the reliance of projects on Stale bot to deal with their PR backlog, the impact of Stale bot on pull-based development, and the kind of PRs usually intervened by Stale bot.
- To promote the reproducibility of our study and facilitate future research on this topic, we publicly share our dataset online at https://doi.org/10.5281/zenodo.7978381.

4.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 4.2 overviews the dataset used for our study. Then, Sections 4.3 to 4.5 report our approach and findings to answer each research question, and Section 4.6 discusses the implications of our study. Finally, Section 4.7 describes the limitations of our study, and Section 4.8 concludes this chapter with a summary of our study.

4.2 Dataset

For our study, we need large and popular open-source projects as their higher workload makes them more likely to benefit significantly from the adoption of Stale bot. The projects should also have a rich history of using Stale bot as part of their pull-based development workflow to ensure that we have enough data for our analyses. To identify such projects, we rely on GH Archive [67], which archives all the public events happening on GitHub [111]. First, we query the GH Archive's public dataset on Google BigQuery [112] and look for all the events performed by Stale bot on PRs (i.e., the actor name is `stale[bot]` and the payload string contains `"pull_request"`). Then, we use the retrieved events to identify the projects that have ever used Stale bot in their pull-based development workflow. Next, we collect the timeline of activities for the identified projects using the `PyGithub` package [62] on November 17th, 2021. The timeline of activities of PRs is provided by the GitHub API [65] and includes the details (e.g., type, actor, and time) of all the events (e.g., commits, comments, labelings, and resolutions) that happened during their lifecycle [113, 64, 63].

We then determine the adoption time of each project by looking for the first event performed by Stale bot in the PRs of the project. Following the recommendation of Wagner et al. [27], we consider 12 months before and 12 months after the adoption (a total of two years) as our observation period. This period allows

Table 4.1: Overview of the projects selected to study the helpfulness of Stale bot for pull-based development.

Project	PRs	Stars	Contributors	Maintainers	Age in Months	Months Since Adoption	Domain	Language
nixos/nixpkgs	121,998	8,013	4,629	300	111	17	Package Manager	Nix
homebrew/homebrew-core	84,686	10,203	6,747	50	67	54	Package Manager	Ruby
ceph/ceph	43,885	9,826	1,573	73	120	35	Storage Platform	C++
automattic/wp-calypso	38,301	11,922	707	151	70	29	WordPress Frontend	JavaScript
home-assistant/core	34,922	47,437	3,875	51	96	19	Home Automation	Python
cleverraven/cataclysm-dda	33,239	5,762	1,686	36	107	26	Video Game	C++
homebrew/linuxbrew-core	23,259	1,171	285	13	64	47	Package Manager	Ruby
istio/istio	21,236	28,423	1,053	78	58	15	Microservice Mesh	Go
qgis/qgis	19,587	5,092	569	39	124	39	GIS System	C++
devexpress/devextreme	19,412	1,521	152	21	53	18	Web Framework	JavaScript
grafana/grafana	18,538	44,786	2,259	55	93	21	Visualization Platform	TypeScript
wikia/app	18,293	191	217	64	105	34	Wiki Engine	PHP
helm/charts	18,196	15,305	4,958	16	69	34	Kubernetes Apps	Go
grpc/grpc	17,939	32,361	1,221	49	81	25	RPC Framework	C++
solana-labs/solana	17,805	5,351	275	30	44	25	Decentralized Blockchain	Rust
home-assistant/home-assistant.io	17,780	2,253	4,410	28	81	46	Home Automation	HTML
conda-forge/staged-recipes	16,119	487	2,443	32	72	19	Package Manager	Python
apache/beam	15,924	5,068	961	33	68	38	Data Pipelines	Java
frappe/erpnext	15,840	9,970	580	42	122	39	ERP System	Python
riot-os/riot	14,335	3,985	447	33	104	26	Operating System	C

us to ensure that we have a sufficient number of observations for our analyses and to take into account the expected seasonal variations in the projects [27]. Therefore, we focus on the projects that have at least 12 months of pull-based development history before the adoption and that have also used Stale bot for at least 12 consecutive months in their pull-based development workflow after the adoption. Among these projects, we finally select the top 20 with the most PRs to focus on the largest and most popular projects.

Table 4.1 provides an overview of the projects that we selected for our study. In summary, the selected projects have thousands of PRs (median of 18,975), thousands of stars (median of 6,888), hundreds of contributors (median of 1,137), tens of maintainers (median of 41), years of pull-based development history (median of 81 months), and years of using Stale bot (median of 28 months). Additionally, these projects span multiple application domains and programming languages, providing a more diverse selection of projects for our study.

4.3 RQ1: How much do the studied projects use Stale bot to deal with their PR backlog?

Open-source projects on GitHub are adopting Stale bot to automatically follow up on the status of their inactive PRs, warn the contributors and maintainers about the lack of activity, and eventually close such PRs if there is no further progress on them [11, 13, 12]. As our first research question, we aim to understand the

extent to which large open-source projects rely on Stale bot to automatically deal with their unresolved PRs. In the following, we first explain our approach and then discuss our findings to answer this research question.

4.3.1 Approach

To investigate the usage of Stale bot, we analyze both its configuration and activity during its first year of adoption (i.e., our observation period) in the studied projects. For each project, we extract the configured number of days of inactivity before a PR is marked as stale (i.e., `daysUntilStale`) and the configured number of days of inactivity before a PR already marked as stale gets closed (i.e., `daysUntilClose`) from its configuration file of Stale bot (i.e., `.github/stale.yml`). We also measure the monthly activity of Stale bot in the PRs of each project as a proxy for the extent to which the project actually relies on Stale bot to deal with its unresolved PRs. Since Stale bot can either warn about the lack of activity in a PR or close a PR due to inactivity, we further distinguish between these two types of activities. For this purpose, we use the following definitions to classify Stale bot activities in a PR:

- **Intervention:** Any commenting, labeling, unlabeling, or closing event performed by Stale bot.
- **Warning:** Any commenting or labeling event performed by Stale bot that is not immediately followed by a closing event from Stale bot within a minute (to account for the processing delay between the closing comment and the actual closure of a PR on GitHub).
- **Closure:** A closing event performed by Stale bot.

4.3.2 Findings

Table 4.2 provides an overview of the usage of Stale bot during its first year of adoption across the studied projects, indicating (1) the number of open PRs immediately upon the adoption (Backlog), (2) the average number of open PRs at the beginning of each month (# Open PRs), (3) the average ratio of open PRs intervened by Stale bot in each month (% Intervened PRs), (4) the average ratio of open PRs warned by Stale bot in each month (% Warned PRs), (5) the average ratio of open PRs closed by Stale bot in each month (% Closed PRs), (6) the average configured number of days to stale a PR in each month (Days to Stale), and (7) the average configured number of days to close a stale PR in each month (Days to Close). Note that the ratio of intervened PRs might not equal the sum of the ratios of warned PRs and closed PRs in a project. This

Table 4.2: Usage of Stale bot during its first year of adoption across the studied projects.

Usage Level	Project	Backlog	Monthly Average # Open PRs	% Intervened PRs	% Warned PRs	% Closed PRs	Days to Stale	Days to Close
High	solana-labs/solana	27	28	70.8%	65.5%	30.5%	15	7
	grafana/grafana	139	109	61.3%	48.4%	11.0%	14	30
	istio/istio	151	160	60.7%	47.3%	12.1%	14	27
	homebrew/linuxbrew-core	174	58	58.4%	51.8%	37.3%	21	7
	helm/charts	423	366	54.8%	43.8%	23.0%	30	14
	frappe/erpnext	14	44	51.0%	46.9%	13.7%	22	7
Moderate	qgis/qgis	40	31	48.0%	41.9%	18.4%	14	7
	wikia/app	23	24	46.1%	41.6%	19.8%	30	7
	homebrew/homebrew-core	72	82	32.5%	28.7%	9.0%	21	7
	conda-forge/staged-recipes	543	226	29.7%	16.7%	13.0%	150	30
	home-assistant/core	228	270	29.0%	26.8%	7.9%	65	7
Low	grpc/grpc	332	207	22.2%	17.3%	11.5%	150	6
	ceph/ceph	742	637	20.2%	15.0%	3.7%	60	90
	apache/beam	127	95	15.4%	13.2%	7.7%	60	7
	cleverraven/cataclysm-dda	65	95	13.2%	10.3%	1.6%	30	30
	riot-os/riot	555	454	11.0%	7.0%	3.7%	185	31
	devexpress/devextreme	31	31	10.6%	9.5%	3.9%	30	5
	home-assistant/home-assistant.io	53	65	7.6%	6.5%	3.7%	60	7
	nixos/nixpkgs	2,054	2,324	6.9%	5.3%	0.0%	180	–
	automattic/wp-calypso	293	349	2.9%	2.5%	1.8%	270	7

Dark gray highlights activities in more than 50% of monthly open PRs and light gray highlights activities in between 25% and 50% of monthly open PRs.

discrepancy arises because intervention includes other activities (i.e., unlabeling events) aside from warnings and closures, and also because PRs might get intervened multiple times within a month (e.g., closure after a warning).

For easier comparison, we group the projects based on the average monthly ratio of open PRs intervened by Stale bot (i.e., % Intervened PRs) into three levels of usage: (1) six projects have a high usage level where Stale bot intervened in more than 50% of monthly open PRs; (2) five projects have a moderate usage level where Stale bot intervened in between 25% and 50% of monthly open PRs; and (3) nine projects have a low usage level where Stale bot intervened in less than 25% of monthly open PRs. In the following, we discuss our findings in more detail.

The usage level of Stale bot widely varies among the projects. On average, Stale bot intervened between 2.9% and 70.8% of open PRs, warned between 2.5% and 65.5% of open PRs, and closed between 0% and 37.3% of open PRs each month in the projects. For example, in the nixos/nixpkgs project, which has both the largest PR backlog upon the adoption and the highest average monthly number of open PRs after the adoption among the studied projects, Stale bot only intervened in 6.9% of open PRs on average each month, and was not even configured to close any of them. In contrast, the solana-labs/solana project has the second lowest average monthly number of open PRs after the adoption among the studied projects, yet Stale bot warned 65.5% of open PRs and closed 30.5% of them on average each month. This wide range of activities indicates that while some projects rely conservatively on Stale bot, others rely on it aggressively to deal with

their unresolved PRs.

The projects with a larger backlog of unresolved PRs have typically not relied more aggressively on Stale bot. The average monthly ratio of open PRs intervened by Stale bot is not significantly associated with the size of the PR backlog upon the adoption (Spearman's $\rho = -0.31$). Similarly, it is not significantly associated with the average monthly number of open PRs after the adoption (Spearman's $\rho = -0.38$). These results suggest that the higher activity of Stale bot in a project is not usually due to more accumulated unresolved PRs. Indeed, there is a significant negative correlation between the average monthly ratio of open PRs intervened by Stale bot and the average configured number of days to mark a PR as stale (Spearman's $\rho = -0.79$), suggesting that the higher activity of Stale bot in a project is usually due to a more aggressive configuration (i.e., fewer days of inactivity before a PR is marked as stale). In other words, the differences in the activity of Stale bot in different projects tend to be due to its configuration rather than the PR backlog size in a project.

Answer to RQ1. We find that the usage level of Stale bot widely varies among the studied projects. On average each month, Stale bot intervened in less than 25% of open PRs in nine projects, between 25% and 50% of open PRs in five projects, and more than 50% of open PRs in six projects. Unexpectedly, the projects with a larger backlog of unresolved PRs have typically not relied more aggressively on Stale bot to deal with their unresolved PRs.

4.4 RQ2: What is the impact of Stale bot on pull-based development in the studied projects?

In the previous research question, we found that the studied projects rely on Stale bot to deal with their accumulated backlog of unresolved PRs. As our second research question, we aim to understand if and how adopting Stale bot improves the efficiency and effectiveness of the pull-based development workflow in large open-source projects. In the following, we first explain our approach and then discuss our findings to answer this research question.

4.4.1 Approach

To investigate the impact of adopting Stale bot, we first need to quantify the performance of the pull-based development workflow in the studied projects. For this purpose, we consult similar studies that investigated the effect of an intervention on the pull-based development of open-source projects [34, 35]. As shown in Table 4.3, we measure 13 performance indicators covering six dimensions: (1) resolved PRs, (2) review latency, (3) resolution time, (4) review discussion, (5) PR updates, and (6) contributor retention. Similar to previous studies [34, 35], we differentiate between the indicators of merged PRs (ending with _m) and closed PRs (ending with _c) to take into account the inherent differences in the characteristics of accepted and rejected PRs. To identify merged PRs, we resort to heuristics [68] similar to [60] because accepted PRs are not always merged using the standard methods provided through the GitHub interface. Accordingly, besides PRs with an explicit merged status (i.e., merged using the GitHub interface), we consider closed PRs with a commit inside the project that references them (e.g., "Close #123") as merged. For each month before and after the adoption, we then measure the indicators of each project by aggregating the data about PRs that have been merged or closed in that month's timeframe.

To estimate the potential impact of Stale bot on the indicators of the projects, we rely on interrupted time-series analysis (ITS) [27]. This method is a well-established quasi-experiment previously used in the software engineering literature to study the impact of an intervention (e.g., [32, 34, 35]). To estimate the longitudinal impact of an intervention (i.e., the adoption of Stale bot in our case), ITS compares a period before and after the intervention assuming the trend would be retained if the intervention had not occurred. To perform ITS in our study, we use the following linear regression model:

$$Y_t = \beta_0 + \beta_1 \times time_t + \beta_2 \times adoption_t + \beta_3 \times time\_since\_adoption_t + \beta_4 \times controls \tag{4.1}$$

where Y_t represents the value of indicator Y at time t; *time* represents the number of months passed since the start of the observation period at time t (encoded from 1 to 24 covering one year before and one year after the adoption); *adoption* represents whether Stale bot has been adopted at time t (encoded as 0 before the adoption and as 1 after the adoption); *time_since_adoption* represents the number of months passed since the adoption of Stale bot at time t (encoded as 1 to 12 covering one year after the adoption and as 0 before the adoption); and *controls* includes a set of variables to control for the inherent differences among the studied projects. These control variables include: (1) the age of the project at the adoption time in months (denoted

as *age_at_adoption*) as a proxy for the level of maturity in the project, (2) the number of PRs at the adoption time in the project (denoted as *pulls_at_adoption*) as a proxy for the level of activity in the project, (3) the number of contributors at the adoption time in the project (denoted as *contributors_at_adoption*) as a proxy for the size of the project community, and (4) the number of maintainers at the adoption time in the project (denoted as *maintainers_at_adoption*) as a proxy for the size of the project core team.

To implement the ITS regression in Equation (4.1), we build linear mixed-effects models [114, 115] using the `lmerTest` package [116]. To do so, we consider *age_at_adoption*, *pulls_at_adoption*, *contributors_at_adoption*, and *maintainers_at_adoption* as fixed effects and the name of the project (denoted as *project_name*) as the random intercept. While both these fixed-effect and random-effect variables aim to capture project-to-project variability, the random intercept allows for capturing unmeasured variability between the projects by assigning a different intercept to each project. We also log-transform all the variables with a skewed distribution to better satisfy the assumptions of linear mixed-effects models, such as linearity and homoscedasticity [114]. To evaluate the goodness of fit of our models, we measure the marginal and conditional coefficients of determination (R^2) [117] using the `performance` package [118]. While the marginal R^2 describes the proportion of the total variance explained only by the fixed effects, the conditional R^2 describes the proportion of the total variance explained by both the fixed and the random effects. To estimate the statistical significance of the coefficients in our models, we rely on the Satterthwaite's method [119] calculated using the `lmerTest` package [116]. We consider the traditional confidence level of 95% (i.e., $\alpha = 0.05$) [120] to identify significant variables. Additionally, to estimate the effect size of each variable, we measure its sum of squares based on type III analysis of variance (a.k.a. ANOVA) [121] using the `lmerTest` package [116]. The results describe the fraction of the total variance explained by the model that can be attributed to each variable. To determine if the adoption of Stale bot has a potential impact on an indicator, we check the statistical significance of *time*, *adoption*, and *time_since_adoption* in the corresponding model. Specifically, a significant *time* shows the existing trend before the adoption, a significant *adoption* shows the change in level at the adoption time, and a significant *time_since_adoption* shows the change in the trend after the adoption. Accordingly, the sum of *time* and *time_since_adoption* represents the new trend after the adoption. To also estimate the effect size of the adoption when a significant change is observed, we rely on counterfactuals [27]: what would have happened if the adoption had never occurred and thus the trend before the adoption had continued unchanged. For this purpose, we assume that there has been neither a change in the level (i.e., *adoption* = 0) nor the trend (i.e., *time_since_adoption* = 0) and measure the change percentage compared to the actual predicted values.

Table 4.3: Overview of the indicators measured to quantify the performance of the pull-based development workflow in the studied projects.

Dimension	Rationale	Indicator	Description
Resolved PRs	Stale bot closes inactive PRs and frees maintainers from manually tracking inactive PRs.	merged_pulls closed_pulls	Number of merged PRs within a month Number of closed PRs within a month
Review Latency	Stale bot frees maintainers from triaging inactive PRs by automatically tracking, warning, and closing such PRs.	first_latency_m first_latency_c	Average first response latency (excluding Stale bot) of merged PRs within a month in hours Average first response latency (excluding Stale bot) of closed PRs within a month in hours
		mean_latency_m mean_latency_c	Average of the mean response latency (excluding Stale bot) of merged PRs within a month in hours Average of the mean response latency (excluding Stale bot) of closed PRs within a month in hours
Resolution Time	Stale bot prevents PRs from staying indefinitely open without any progress by closing inactive PRs.	resolution_time_m resolution_time_c	Average resolution time of merged PRs within a month in hours Average resolution time of closed PRs within a month in hours
Review Discussion	Stale bot attracts the attention of participants by warning about the lack of activity.	comments_m comments_c	Average number of comments (excluding Stale bot) in merged PRs within a month Average number of comments (excluding Stale bot) in closed PRs within a month
PR Updates	Stale bot prevents PRs from getting too outdated by warning about the lack of activity.	commits_m commits_c	Average number of commits in merged PRs within a month Average number of commits in closed PRs within a month
Contributor Retention	Stale bot is frequently reported to frustrate the community by attempting to close PRs.	contributors	Number of active contributors within a month

4.4.2 Findings

The results of the models to estimate the impact of Stale bot on different performance indicators are presented in Section 6.2.5. For easier comparison, Figure 4.2 visualizes how the values of the impacted indicators vary each month during our observation period, along with the average of the model predictions (the solid red lines) and the average of the counterfactual predictions for all the studied projects (the dashed red lines). The variation plots for the remaining indicators can also be found in Section 6.2.5. We observe that the projects closed more PRs within the first few months of adopting Stale bot, but overall closed and merged fewer PRs afterward. The adoption of Stale bot is also associated with faster first reviews in merged PRs, faster resolutions in closed PRs, slightly fewer updates in merged PRs, and considerably fewer active contributors in the projects. In the following, we discuss our findings in more detail.

While more PRs are closed within the first few months of adopting Stale bot, overall fewer PRs are closed and merged afterward. As shown in Figure 4.2b, the number of closed PRs experienced an increase in the level but a decrease in the slope that reversed the increasing trend before the adoption. Specifically, our predictions indicate that 15% more PRs were closed in the first month of adoption but overall 10% fewer PRs were closed by the end of the first year of adoption. The short-term increase in the number of monthly closed PRs is expected as Stale bot closes accumulated inactive PRs immediately upon its adoption. From Figure 4.2a, we also observe a decrease in both the level and the slope of the number of merged PRs that decelerated the increasing trend before the adoption. While more PRs are still merged each month, our predictions indicate that overall 24% fewer PRs were merged by the end of the first year of adoption.

Merged PRs tend to have faster first reviews after the adoption of Stale bot. As shown in Figure 4.2c, the first review latency of merged PRs experienced a decrease in the slope that reversed the increasing trend before the adoption. Specifically, our predictions indicate that merged PRs had an overall 21% lower first review latency during the first year of adoption. However, the first review latency of closed PRs and the mean review latency of both closed PRs and merged PRs have not significantly changed after the adoption. Still, closed PRs tend to take longer to receive a first review over time, and this trend has not significantly changed after the adoption. The results suggest that the maintainers are focusing on PRs that are more likely to get merged, but at the cost of leaving the remaining PRs to linger until Stale bot closes them after a period of inactivity.

Closed PRs tend to have faster resolutions after the adoption of Stale bot. As shown in Figure 4.2d,

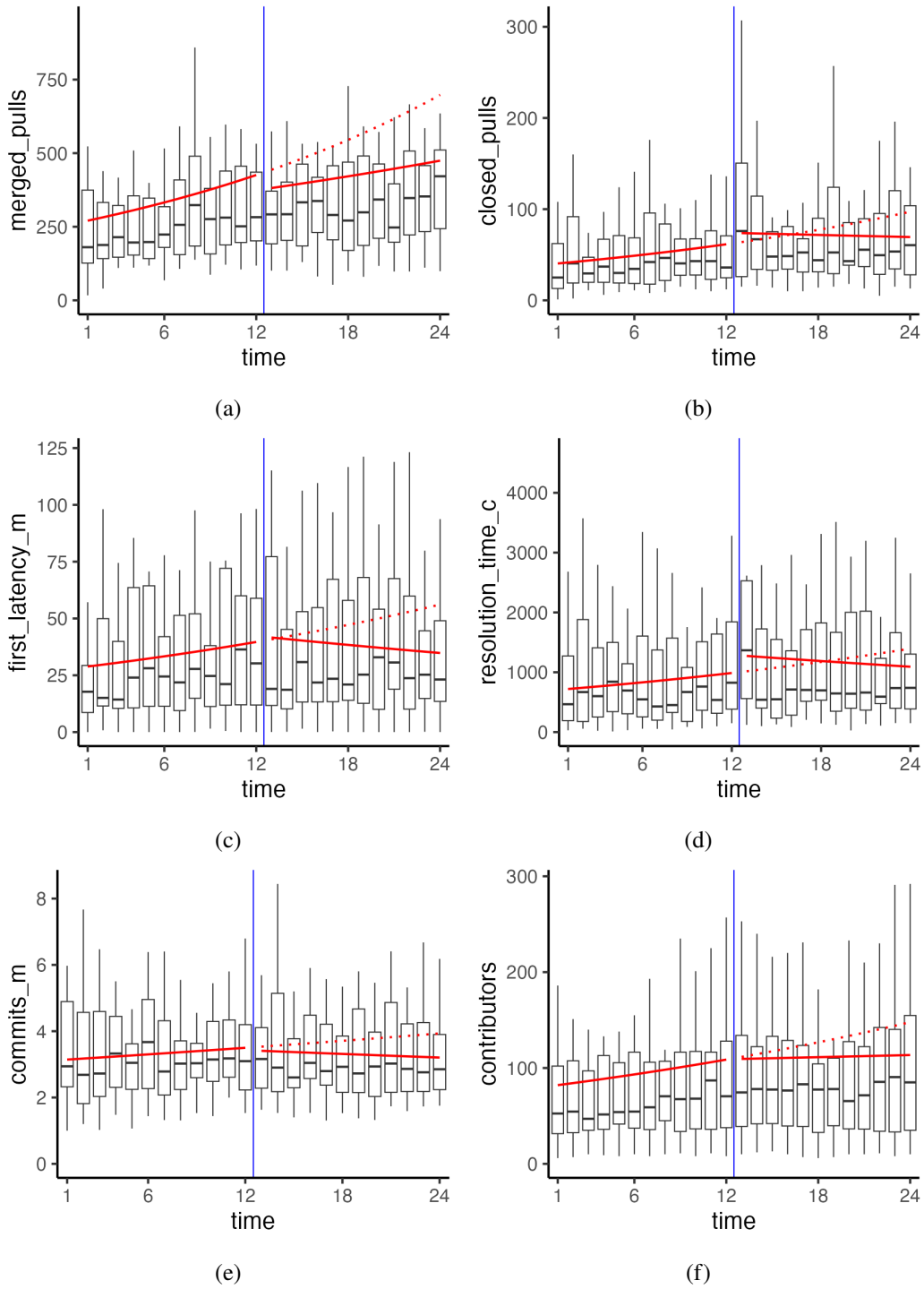


Figure 4.2: Variation in (a) the number of merged PRs, (b) the number of closed PRs, (c) the first response latency of merged PRs, (d) the resolution time of closed PRs, (e) the number of commits in merged PRs, and (f) the number of active contributors each month during our observation period. The blue lines show the adoption time, the solid red lines show the average predictions, and the dashed red lines show the average counterfactual predictions of the model for all the studied projects. Note that we dropped the outliers in the plots and used the exponential function to convert the log-transformed output of the models back to real values.

the resolution time experienced a decrease in the slope across closed PRs that reversed the increasing trend before the adoption, but has not significantly changed across merged PRs. For example, our predictions indicate that closed PRs had 22% lower resolution time in the last month of the first year of adoption. The decrease in the resolution time of closed PRs is expected as the auto-close function of Stale bot does not allow PRs to stay open indefinitely without any activity.

The amount of discussions in PRs is not affected by the adoption of Stale bot. We find that the number of discussion comments has not significantly changed after the adoption of Stale bot across both merged PRs and closed PRs. This observation comes as a surprise since we expected Stale bot to encourage more communication during the review process of PRs by notifying the participants about the lack of activity and by reducing the workload of the maintainers.

Merged PRs tend to have slightly fewer updates after the adoption of Stale bot. As shown in Figure 4.2e, the total number of commits across merged PRs experienced a decrease in the slope but has not significantly changed across closed PRs. Specifically, our predictions indicate that merged PRs overall contained 11% fewer commits during the first year of adoption. Still, closed PRs tend to have more commits over time, and this trend has not significantly changed after the adoption.

The adoption of Stale bot is associated with a considerable decrease in the number of active contributors. As shown in Figure 4.2f, the number of monthly active contributors experienced a decrease in the slope that significantly decelerated the increasing trend before the adoption. Specifically, our predictions indicate that overall 14% fewer contributors were active each month during the first year of adoption. The decreased number of active contributors may reflect the frustration of the community regarding the usage of Stale bot in open-source projects.

Answer to RQ2. We find that the studied projects closed more PRs within the first few months of adopting Stale bot, but overall closed and merged fewer PRs afterward. The adoption of Stale bot is also associated with faster first reviews in merged PRs, faster resolutions in closed PRs, slightly fewer updates in merged PRs, and considerably fewer active contributors in the projects.

Table 4.4: Overview of the factors measured to characterize PRs, their contributors, and their review processes.

Dimension	Factor	Description
Pull Request	description	Number of words in the title and description of the PR
	initial_commits	Number of commits at the submission time of the PR
	followup_commits	Number of commits after the submission time of the PR
	initial_changed_lines	Number of changed lines at the submission time of the PR
	followup_changed_lines	Number of changed lines after the submission time of the PR
	initial_changed_files	Number of changed files at the submission time of the PR
	followup_changed_files	Number of changed files after the submission time of the PR
Contributor	submitted_pulls	Number of previously submitted PRs by the contributor of the PR in the project
	acceptance_rate	Ratio of the previously merged PRs of the contributor of the PR in the project
	contribution_period	Number of months since the first submitted PR of the contributor of the PR in the project
Review Process	participants	Number of participants (excluding Stale bot) during the lifecycle of the PR
	participant_comments	Number of comments from the participants (excluding Stale bot) during the lifecycle of the PR
	contributor_comments	Number of comments from the contributor during the lifecycle of the PR
	first_latency	Number of hours till the first response of the participants (excluding Stale bot) in the PR
	mean_latency	Mean number of hours between the responses of the participants (excluding Stale bot) in the PR
	resolution_time	Number of hours between the submission and resolution times of the PR

4.5 RQ3: What kind of PRs are usually intervened by Stale bot in the studied projects?

In the previous research question, we found that the adoption of Stale bot is associated with both positive and negative changes in the pull-based development workflow of the studied projects. As our last research question, we aim to understand what characteristics of PRs, their contributors, and their review processes are associated with a higher probability of getting intervened by Stale bot in large open-source projects. In the following, we first explain our approach and then discuss our findings to answer this research question.

4.5.1 Approach

To investigate the kinds of PRs intervened by Stale bot, we first need to characterize the PRs, their contributors, and their review processes. For this purpose, we consult the pull-based development literature [23, 22, 72, 71]. As shown in Table 4.4, we measure 16 factors covering three dimensions: (1) PR characteristics, (2) contributor characteristics, and (3) review process characteristics. After measuring these factors, we perform statistical analyses to determine how each factor differs in PRs that are intervened by Stale bot. To have a fair comparison between intervened and not intervened PRs, we filter our dataset to include only PRs that are closed or merged within the first year of adoption (i.e., our observation period). After this step, our dataset includes a total of 126,378 PRs, of which 6,833 PRs (~5.4%) have been intervened by Stale bot.

Then, we compare the distribution of the measured factors between intervened and not intervened PRs by generating violin plots [73] for each project using the `ggstatsplot` package [74]. The generated plots for each factor showing their median values (denoted by M), interquartile ranges (the box inside the violin), and probability densities (the width of the violin at each value) are presented in Section 6.2.5. To test the statistical difference between the factors of intervened and not intervened PRs, we apply the MannWhitney U test [75] as a nonparametric test that does not require the distribution of the factors to be normal. To perform this test, we use the `stats` package [76] and add the results to the generated plots. We consider the traditional confidence level of 95% (i.e., $\alpha = 0.05$) [120] to identify statistically significant factors.

While statistical significance verifies whether a difference exists between the factors of intervened and not intervened PRs, we also need to test their practical difference [77]. For this purpose, we use Cliff's delta [78] to estimate their magnitude of difference (i.e., effect size). The value of Cliff's delta (denoted by d) ranges from -1 to $+1$, where a positive d implies that the values of the factor in intervened PRs are usually greater than those of not intervened PRs, while a negative d implies the opposite. To calculate this statistic, we use the `effectsize` package [79] and add the results to the generated plots. Finally, for easier comparison, we convert the d values to qualitative magnitudes according to the following thresholds as suggested by Hess and Kromrey [80]:

$$\text{Effect size} = \begin{cases} \text{Negligible}, & \text{if } |d| \leq 0.147 \\ \text{Small}, & \text{if } 0.147 < |d| \leq 0.33 \\ \text{Medium}, & \text{if } 0.33 < |d| \leq 0.474 \\ \text{Large}, & \text{if } 0.474 < |d| \leq 1 \end{cases}$$

4.5.2 Findings

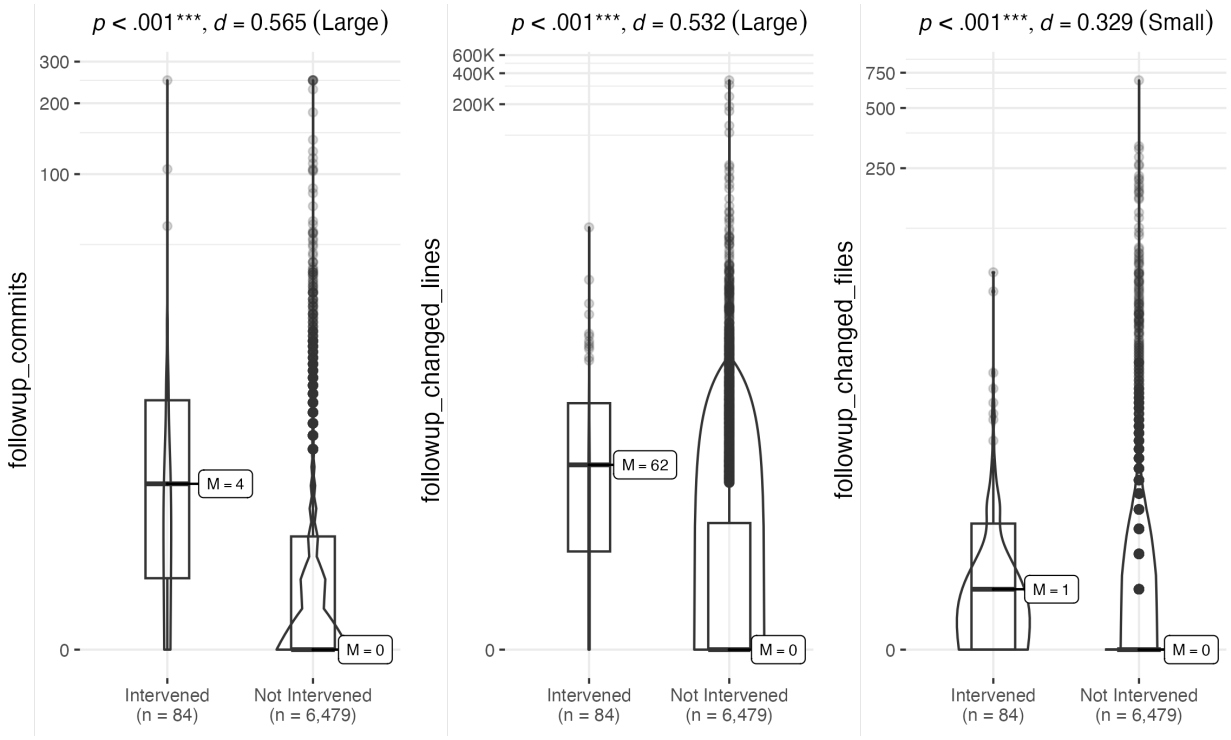
Table 4.5 summarizes the significant differences in the characteristics of PRs with and without intervention from Stale bot across the studied projects. For each factor, the table shows the number of projects in which intervened PRs tend to have significantly higher values compared to not intervened PRs, the number of projects in which intervened PRs and not intervened PRs are not significantly different, and the number of projects in which intervened PRs tend to have significantly lower values compared to not intervened PRs. We consider a factor significant if its difference between intervened and not intervened PRs is both statistically

Table 4.5: Differences in the characteristics of PRs with and without intervention from Stale bot across the studied projects.

Dimension	Factor	Higher in Intervened PRs			No Difference	Lower in Intervened PRs		
		Small	Medium	Large		Small	Medium	Large
Pull Request	description	4	1	1	10	3	–	1
	initial_commits	–	–	–	12	7	–	1
	followup_commits	8	4	2	6	–	–	–
	initial_changed_lines	1	–	–	16	2	–	1
	followup_changed_lines	8	4	2	6	–	–	–
	initial_changed_files	–	–	–	13	6	–	1
	followup_changed_files	9	2	–	9	–	–	–
Contributor	submitted_pulls	–	–	–	–	3	14	3
	acceptance_rate	–	–	–	1	6	11	2
	contribution_period	–	–	–	2	12	5	1
Review Process	participants	5	–	8	4	2	1	–
	participant_comments	4	6	4	3	–	2	1
	contributor_comments	5	6	3	6	–	–	–
	first_latency	2	5	6	4	2	–	1
	mean_latency	–	2	17	1	–	–	–
	resolution_time	–	–	20	–	–	–	–

significant (i.e., $p < 0.05$) and practically significant (i.e., the effect size is small, medium, or large). We observe the most common differences in the characteristics of contributors and the review processes of PRs. Specifically, the largest differences are in the resolution time of PRs (higher in all the projects), the number of prior PRs by contributors (lower in all the projects), the mean response latency of PRs (higher in 19 projects), the acceptance rate of contributors (lower in 19 projects), and the contribution period of contributors (lower in 18 projects). In the following, we discuss our findings in more detail.

PRs intervened by Stale bot tend to be more complex. As shown in the PR dimension of Table 4.5, intervened PRs tend to significantly include fewer commits (8 projects), contain smaller changes (3 projects), and touch fewer files (7 projects) upon submission compared to not intervened PRs. However, after the submission, intervened PRs tend to significantly include more commits (14 projects), contain larger changes (14 projects), and touch more files (11 projects). For example, the intervened PRs in the cleverraven/cataclysm-dda project on median have 4 more commits, 62 more changed lines, and 1 more changed file after their submission (see Figure 4.3). Regarding the description length, we also observe that 6 projects tend to have lengthier descriptions and 4 projects tend to have shorter descriptions in intervened PRs. While the description length of PRs rarely changes during the review process of a PR, the size of changes (i.e., the number of commits, changed lines, or changed files) is likely to increase as the PR gets iteratively reviewed and updated.

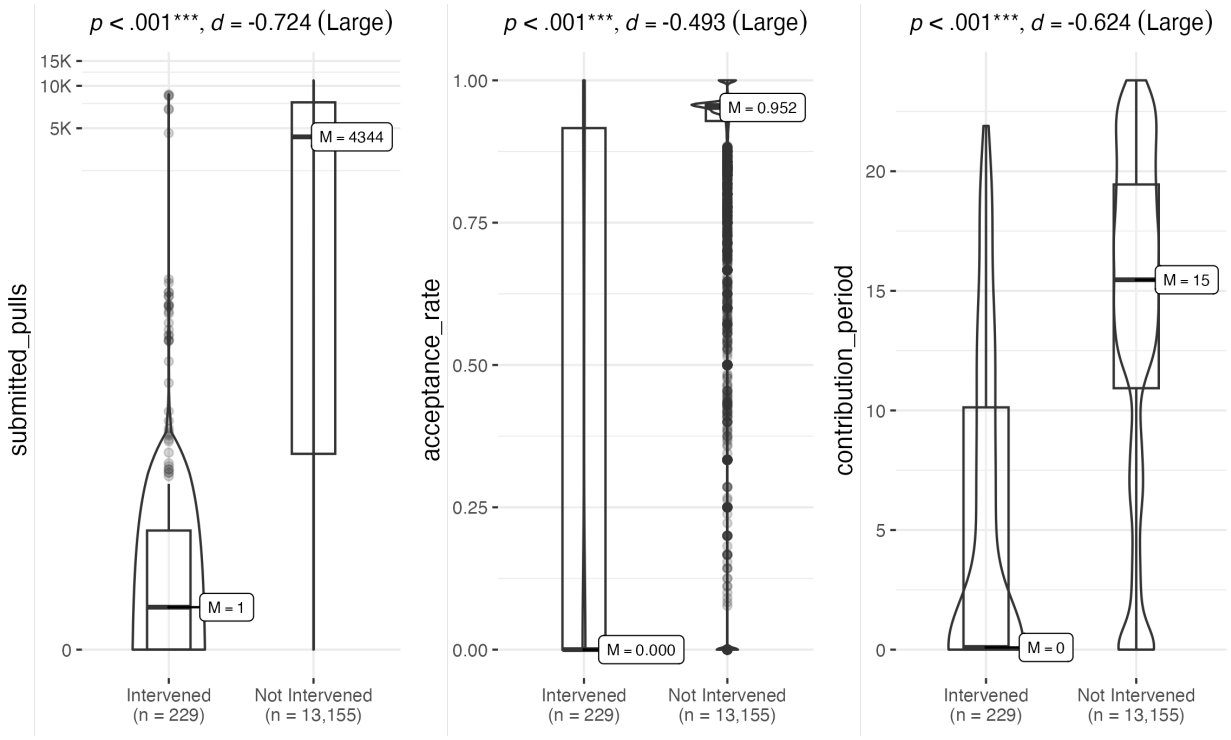


(a) (b) (c)

Figure 4.3: Differences between PRs with and without intervention from Stale bot in the cleverraven/cataclysm-dda project regarding (a) the number of follow-up commits, (b) the number of follow-up changed lines, and (c) the number of follow-up changed files after the submission.

The results indicate that PRs mostly intervened by Stale bot have received considerable effort from their contributors to make it ready to get merged into the project.

The contributors of PRs intervened by Stale bot tend to be less experienced. As shown in the contributor dimension of Table 4.5, the contributors of intervened PRs tend to significantly have previously submitted fewer PRs (all the projects), have a lower acceptance rate (19 projects), and have a lower contribution period (18 projects) compared to the contributors of not intervened PRs. For example, the contributors of intervened PRs in the homebrew/homebrew-core project on median have previously submitted 4,343 fewer PRs, have a 0.95 lower rate of acceptance, and have contributed to the project for 15 fewer months (see Figure 4.4). However, novice contributors are the ones who face the most barriers and thus need the most guidance from the maintainers [109, 105, 104]. The lack of responsiveness from the reviewers is also cited as a major reason why contributors (especially novice or casual contributors) leave the review processes of PRs unfinished and even stop further contributing to the project [24, 8, 107, 104]. Therefore, the results indicate that while Stale bot can help projects automatically deal with their inactive PRs, it may also lead to lower



(a) (b) (c)

Figure 4.4: Differences between the contributors of PRs with and without intervention from Stale bot in the homebrew/homebrew-core project regarding (a) the number of prior PRs, (b) the acceptance rate, and (c) the contribution period.

engagement of the community and eventually contributor abandonment, especially if the reviewers' input is required to continue the review process.

The review processes of PRs intervened by Stale bot tend to be lengthier. As shown in the review process dimension of Table 4.5, the review processes of intervened PRs tend to significantly involve more participants (13 projects), receive more comments from the participants (14 projects), receive more comments from the contributors (14 projects), take longer to receive their first review (13 projects), take longer to receive a follow-up review (19 projects), and take longer to get resolved (all the projects) compared to the review processes of not intervened PRs. For example, the review processes of intervened PRs in the homebrew/homebrew-core project on median involve 2 more participants, receive 5 more comments from the participants, receive 2 more comments from the contributors, takes 14 more hours to receive their first review, takes 146 more hours to receive a follow-up review, and takes 1,168 more hours to get resolved (see Figure 4.5). While a longer resolution time in intervened PRs is expected as Stale bot intervenes in PRs that have been idle for a while, a longer time to receive reviews (both first and follow-up) from the participants is

rather interesting.

Answer to RQ3. We find that Stale bot tends to intervene more in complex PRs, PRs from novice contributors, and PRs with lengthy review processes. Specifically, besides the resolution time of PRs, the largest differences are observed in the number of prior PRs by contributors, the mean response latency of PRs, the acceptance rate of contributors, and the contribution period of contributors.

4.6 Implications

In the following, we combine our findings to further discuss the implications of our study.

4.6.1 Stale bot can help projects deal with a backlog of unresolved PRs

Almost all the studied projects relied on Stale bot to automatically close their accumulated PRs after a period of inactivity. In fact, the projects closed more PRs within the first few months of adopting Stale bot, yet closed and even merged considerably fewer PRs afterward. These findings suggest that adopting Stale bot could be an effective strategy for quickly dealing with a backlog of unresolved PRs in the short term. However, projects should be aware that the rule-based nature of Stale bot could wrongly close PRs that may still be under progress despite being inactive for some time [18]. Moreover, the automatic closure of PRs by Stale bot is known to raise the most negative reactions from the contributors and participants, especially when they perceive the closure as erroneous or unjustified [19].

4.6.2 Stale bot can help projects improve the review process of PRs

After the adoption of Stale bot, PRs that ended up being merged received faster reviews after their submission, and PRs that ended up being closed were also resolved much faster in the studied projects. However, PRs that end up being closed are increasingly taking longer to receive a review from the reviewers. The adoption of Stale bot did not improve this trend or encourage more communication during the review process of PRs. These findings suggest that maintainers are focusing on PRs that are more likely to get merged, but at the cost of leaving the remaining PRs to linger until Stale bot closes them after a period of inactivity. However, projects should be mindful that such a strategy will likely result in fewer merged PRs over time.

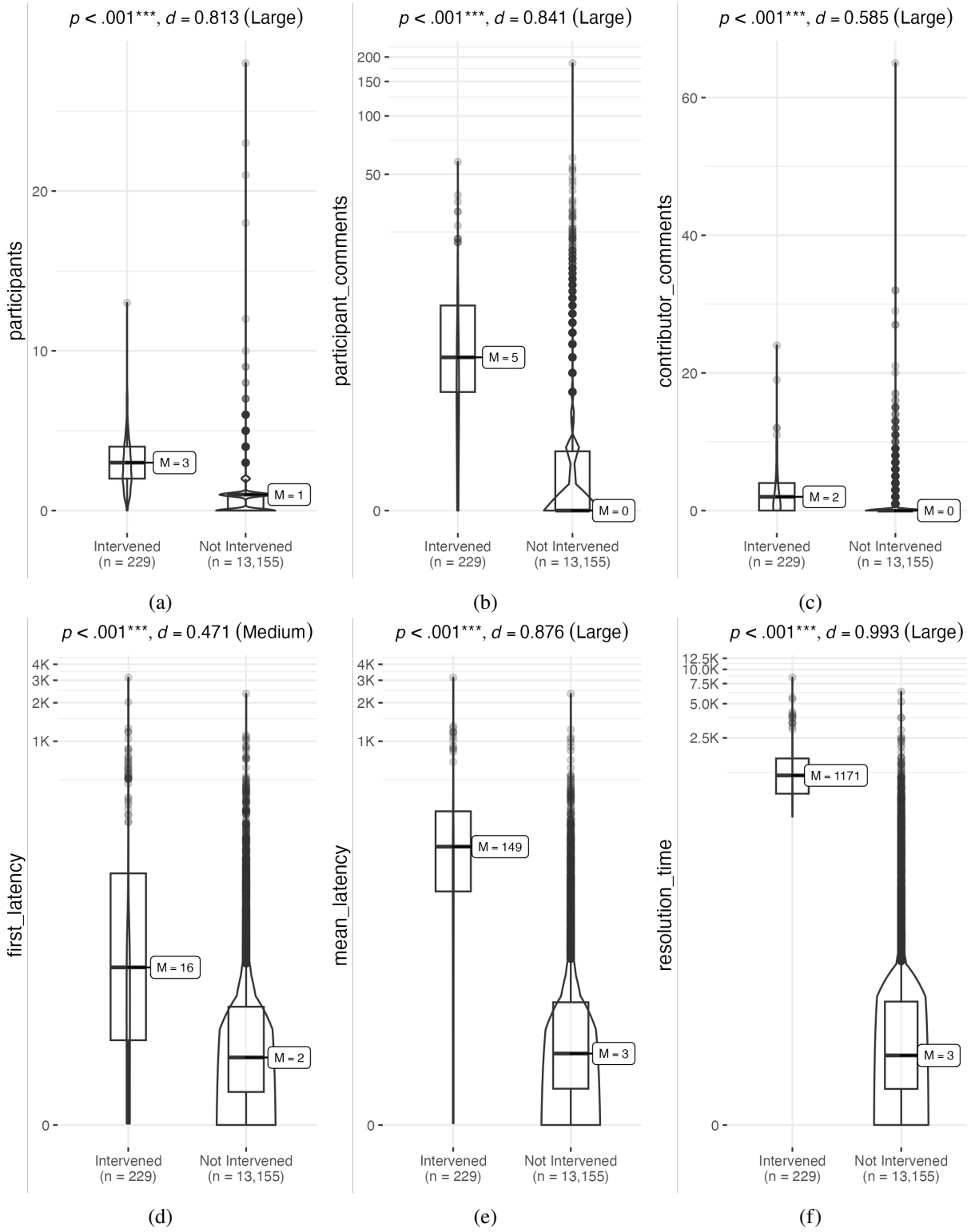


Figure 4.5: Differences between the review processes of PRs with and without intervention from Stale bot in the homebrew/homebrew-core project regarding (a) the number of participants, (b) the number of comments from the participants, (c) the number of comments from the contributors, (d) the first review latency, (e) the mean review latency, and (f) the resolution time.

4.6.3 Stale bot can negatively affect the contributors of projects

The studied projects experienced a considerable decrease in their number of active contributors after the adoption of Stale bot. While Stale bot can help projects automatically deal with their inactive PRs, it is also more likely to intervene in PRs submitted by less experienced contributors. Projects should ensure that PRs receive timely reviews and responses from their reviewers, particularly if their input is needed to continue the review process. This is particularly important as the lack of responsiveness from reviewers is cited as a major reason why contributors (especially novice or casual contributors) leave the review process of PRs unfinished and even stop further contributing to a project [24, 8, 107, 104]. Therefore, implicitly ignoring unresolved PRs till Stale bot eventually closes them may lead to decreased community engagement and an increased probability of contributor abandonment.

4.7 Limitations

In the following, we discuss threats to the internal, construct, and external validity [108] of our study and explain the measures taken to mitigate them.

4.7.1 Construct Validity

Construct validity is concerned with how accurately we quantified the helpfulness of Stale bot for pull-based development. We measured 13 different performance indicators covering various dimensions, including resolved PRs, review latency, resolution time, review discussion, PR updates, and contributor retention. To identify these indicators, we consulted similar studies that investigated the effects of interventions on the pull-based development of projects [34, 35] and also drew from our previous experience studying abandoned PRs [24]. Nevertheless, projects may experience changes in aspects that we did not consider or that are difficult to quantify, such as code quality. Forsgren et al. [122] describes various facets of productivity for individual developers and software teams, which future studies can consult for a more comprehensive investigation of the helpfulness of Stale bot.

4.7.2 Internal Validity

Internal validity is concerned with whether the adoption of Stale bot has actually caused the observed effects. Although we applied interrupted time-series analysis as a well-established quasi-experiment to

understand the impact of adopting Stale bot, we cannot conclusively claim that Stale bot caused the observed changes. It is possible that the adoption of Stale bot in a project occurred simultaneously with other events that could explain the observed changes, such as the introduction of new tools or practices. For example, less than a month before adopting Stale bot, the devexpress/devextreme project modified its continuous integration (CI) workflow to also run on submitted PRs in addition to pushed commits [123]. To mitigate this threat, we utilized mixed-effects models for implementing the interrupted time-series analysis. This approach allows us to identify changes that are commonly observed across the majority of projects rather than changes that are specific to only a few projects. In other words, it is unlikely that the majority of the studied projects experienced significant changes to their pull-based development workflow during the same month they adopted Stale bot.

4.7.3 External Validity

External validity is concerned with how well our findings may be generalized to other open-source projects. In this study, we aimed to evaluate the helpfulness of Stale bot in open-source projects, specifically concerning their pull-based development. To conduct our study, we selected 20 large and popular open-source projects with a rich history of using Stale bot in their pull-based development workflow. While we believe these projects are more likely to benefit from adopting Stale bot, we recognize that they cannot represent the entire open-source ecosystem. In other words, the selected projects may not be representative of other open-source projects with different characteristics, such as their size, maturity, popularity, workload, culture, and development practices. Therefore, the findings of this study may not apply to all open-source projects. To acquire more broadly applicable insights, future research can replicate this study using a more diverse selection of projects.

4.8 Chapter Summary

PRs that are neither progressed nor resolved accumulate over time, clutter the list of PRs, and eventually make it difficult for the maintainers to manage and prioritize unresolved PRs. To automatically track such inactive PRs, follow up on their progress, and close them if needed, Stale bot was introduced by GitHub. Despite its increasing adoption, there are ongoing debates on whether using Stale bot alleviates or exacerbates the problem of inactive PRs. To better understand if and how Stale bot helps open-source projects in their

pull-based development workflow, we conducted an empirical study of 20 large and popular open-source projects. First, we analyzed the configuration and activity of Stale bot to understand the extent to which the projects relied on Stale bot to automatically deal with their unresolved PRs. Then, we applied interrupted time-series analysis as a well-established quasi-experiment to understand if and how adopting Stale bot improved the efficiency and effectiveness of the pull-based development workflow in the projects. Next, we analyzed the characteristics of PRs, their contributors, and their review processes to understand the factors that were associated with a higher probability of getting intervened by Stale bot in the projects. Finally, we combined our observations to discuss the potential benefits and drawbacks of employing Stale bot within a pull-based development workflow. In summary, we found that Stale bot can help projects deal with a backlog of unresolved PRs and also improve the review process of PRs. However, the adoption of Stale bot can negatively affect the contributors (especially novice or casual contributors) in a project.

To help better mitigate PR abandonment, in the next chapter, we propose a machine learning approach that predicts the first response latency of maintainers and contributors of PRs. This awareness can lead to managed expectations for both the maintainers and the contributor and also enables them to take proactive actions to mitigate potential challenges during the review process of a PR before it gets abandoned.

Chapter 5

Predicting the First Response Latency of Maintainers and Contributors in Pull Requests

5.1 Introduction

Pull-based development has become a common paradigm for contributing to and reviewing code changes in numerous open-source projects [1, 2]. PRs are the driving force behind the maintenance and evolution of these projects, encompassing everything from bug fixes to new features. Contributors initiate this collaborative process by submitting a PR that proposes changes for integration into the project. The PR then undergoes a review process, during which the contributor revises the changes based on feedback from the project maintainers. This cycle repeats until the PR satisfies the maintainers' requirements for getting merged [4, 5].

The success of the PR depends on the responsiveness of both the maintainers and the contributor during the review process [24, 8, 26, 9]. Timely responses from the maintainers set a positive tone for the entire review process, increasing the likelihood of the contributor continuing the review process towards completion [24, 5]. Conversely, delayed responses are often perceived as negligence, increasing the risk of the contributor abandoning the PR [24, 8, 107]. Once the maintainers have responded, the contributor's promptness in addressing the feedback is equally crucial. Timely responses help maintain the momentum of the review process, whereas delayed responses can cause it to stale [24, 5, 28].

Knowing the expected waiting times can lead to better interactions and managed expectations for both sides. Contributors, when aware of anticipated waiting times, can adjust their schedules accordingly, reducing uncertainty and preserving their motivation throughout the review process [24, 26]. Maintainers, aware of possible delays in contributor responses, can proactively offer additional support or take action to mitigate potential blockers [24]. This awareness also allows maintainers to better allocate their time and resources and prioritize PR reviews [9]. Furthermore, analyzing response time trends can help projects pinpoint and rectify bottlenecks, thereby enhancing the efficiency and effectiveness of their PR review workflows.

The first responses are of particular importance as they not only directly influence the duration [21, 22] and the outcome [23, 24, 8] of the review process, but also the likelihood of future contributions by the contributor [25, 21]. Despite the critical role of first responses, existing approaches only aim to predict the completion time of PRs [124, 125] or nudge overdue PRs [126]. Our study bridges this gap by proposing a machine learning approach to predict: (1) the first response latency of the *maintainers* following the submission of a PR, and (2) the first response latency of the *contributor* after receiving the first response from the maintainers.

For this purpose, we start by curating a dataset of 20 popular and large open-source projects on GitHub. Next, we extract 21 features to characterize projects, contributors, PRs, and review processes. Using these features, we then evaluate seven types of classifiers to identify the best-performing models. Finally, we perform permutation feature importance [94] and SHAP [127] analyses to understand the importance and impact of different features on the predicted response latencies. In summary, we aim to answer the following four research questions:

RQ1: (Maintainers) How well can we predict the first response latency of maintainers? We find that the CatBoost models outperform other models in predicting the first response latency of maintainers, achieving an average improvement of 29% in AUC-ROC and 51% in AUC-PR compared to a no-skilled classifier across the studied projects.

RQ2: (Maintainers) What are the major predictors of the first response latency of maintainers? We find that PRs submitted earlier in the week, containing an average or slightly above-average number of commits at submission, and with more concise descriptions are more likely to get faster responses. Similarly, contributors with a higher acceptance rate and a history of timely responses in the project tend to obtain quicker responses.

RQ3: (Contributors) How well can we predict the first response latency of contributors? Similar to

the first response latency of maintainers, we find that the CatBoost models outperform other models in predicting the first response latency of contributors, achieving an average improvement of 39% in AUC-ROC and 89% in AUC-PR compared to a no-skilled classifier across the studied projects.

RQ4: (Contributors) What are the major predictors of the first response latency of contributors? We find that contributors of PRs that experienced a lower first response latency from maintainers, PRs that received the first response of maintainers earlier in the week, and PRs containing an average or slightly above-average number of commits till the first response of maintainers are more likely to provide faster responses. Similarly, contributors with a history of timely responses in the project and with a higher acceptance rate tend to give quicker responses.

Finally, we evaluate our approach in a cross-project setting. This is especially useful for new projects with limited historical data to build accurate models. Compared to a no-skilled classifier, the models achieve an average improvement of 33% in AUC-ROC and 58% in AUC-PR for maintainers, as well as an average improvement of 42% in AUC-ROC and 95% in AUC-PR for contributors. Furthermore, we find that the key predictors in the cross-project setting are: submission day, number of commits, contributor acceptance rate, historical maintainers responsiveness, and historical contributor responsiveness for maintainers' first response latency; and first review latency, review day, historical contributor responsiveness, number of commits, and contributor activity within the PR for contributors' first response latency. We believe that by predicting the first response latencies, our approach helps open-source projects facilitate collaboration between maintainers and contributors during the review process of PRs.

5.1.1 Our Contributions

In summary, we make the following contributions in this chapter:

- To the best of our knowledge, we are the first to propose machine learning models for classifying the first response latency of maintainers and contributors.
- We investigate the major predictors of the first response latency of maintainers and contributors and discuss the impact of the features on the anticipated waiting periods.
- To promote the reproducibility of our study and facilitate future research on this topic, we publicly share our scripts and dataset online at https://doi.org/10.5281/zenodo.10119284.

5.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 5.2 overviews the design of our study. Sections 5.3 and 5.4 report the results for predicting the first response latency of maintainers and contributors, respectively. Section 5.5 evaluates our approach in a cross-project setting and Section 5.6 discusses the threats to the validity. Finally, Section 5.7 concludes this chapter.

5.2 Study Design

The main objective of this study is to develop machine learning models for predicting the first response latency of the maintainers following the submission of a PR; as well as the first response latency of the contributor after receiving the first response from the maintainers. Additionally, we aim to identify and discuss the impact of the key features that significantly influence the predicted first response latency of maintainers and contributors. In the following, we explain the methodology and design of our study in detail.

5.2.1 Studied Projects

To ensure that we have enough historical data for our study, we seek large and popular open-source projects with a rich history of pull-based development. For this purpose, we rely on GitHub as a pioneer in supporting pull-based development and the largest open-source ecosystem [59], which has also been the subject of numerous code review studies [128, 129]. To identify such projects, we use the number of stars as a proxy for the popularity of the projects [61] and retrieve the list of the top 1,000 most-starred projects. Among these projects, we select the top 20 with the highest number of PRs to focus on the largest and most popular software projects. Table 5.1 provides an overview of the projects that we selected for our case study. In summary, the selected projects have thousands of PRs (median of 42,630), thousands of stars (median of 52,226), hundreds of contributors (median of 3,240), tens of maintainers (median of 201), and years of pull-based development history (median of 106 months). Additionally, these projects span multiple application domains and programming languages, providing a more diverse selection of projects for our case study. We collected the timeline of activities for the selected projects on December 1st, 2022. The timeline of activities of PRs is provided by the GitHub API [65] and includes the details (e.g., type, actor, and time) of all the events (e.g., commits, comments, and resolutions) that occurred during the lifecycle of a PR [113, 64, 63].

Table 5.1: Overview of the projects selected for our study.

Project	PRs	Stars	Contributors	Maintainers	Age	Domain	Language
Odoo	89,513	27,267	2,678	237	102	Business Management System	JavaScript
Kubernetes	72,001	94,103	5,613	361	101	Container Orchestration System	Go
Elasticsearch	60,582	61,986	3,016	299	153	Data Analytics Engine	Java
PyTorch	60,072	60,605	3,610	350	75	Machine Learning Framework	C++
Rust	58,563	74,949	4,566	135	149	Programming Language	Rust
DefinitelyTyped	53,843	41,717	18,980	548	121	TypeScript Type Definitions	TypeScript
HomeAssistant	48,478	56,269	4,646	913	110	Home Automation System	Python
Ansible	48,262	55,575	7,835	93	128	IT Automation Platform	Python
CockroachDB	45,884	26,127	725	203	105	Database Management System	Go
Swift	45,560	61,197	1,333	230	85	Programming Language	C++
Flutter	39,700	146,697	2,287	236	92	Software Development Kit	Dart
Spark	38,809	34,459	3,463	80	105	Data Analytics Engine	Scala
Python	37,792	49,172	3,779	104	69	Programming Language	Python
Sentry	35,489	32,665	834	170	146	Application Performance Monitoring	Python
PaddlePaddle	32,738	19,230	831	500	75	Machine Learning Framework	C++
Godot	30,886	55,580	2,688	63	106	Game Engine	C++
Rails	30,386	51,852	6,435	114	175	Web Application Framework	Ruby
Grafana	30,373	52,599	2,657	199	107	Data Visualization Platform	TypeScript
ClickHouse	29,820	26,262	1,377	54	77	Database Management System	C++
Symfony	29,009	27,679	4,240	51	154	Web Application Framework	PHP

5.2.2 Identification of First Responses

In this study, we focus on responses from project maintainers, as their deeper understanding of the project enables them to evaluate PRs more effectively. To identify maintainers, we follow an approach similar to [130]. We consider a maintainer as a developer with write permissions. This status not only enables them to perform critical maintenance tasks but also gives them the authority to make decisions on accepting or rejecting PRs. To identify maintainers, we analyze each developer's activity within the project, looking for any of the following events that require privileged access: `added_to_project`, `deployed`, `deployment_environment_changed`, `locked`, `merged`, `moved_columns_in_project`, `removed_from_project`, `review_dismissed`, `unlocked`, and `user_blocked`. Additionally, we consider actions such as merging a PR using other nonstandard methods (e.g., through commit messages that include keywords to resolve a PR, like "resolves #123") or closing someone else's PR as privileged events (since users can close their own PRs without requiring any special access). Developers observed engaging in any of these privileged events are classified as maintainers henceforth.

Similar to [21], we define the first response of maintainers as the first feedback (i.e., `commented`, `reviewed`, `line-commented`, and `commit-commented`) or resolution (i.e., `merged`, `closed`, and `reopened`) by a maintainer (excluding bots) other than the contributor. We also define the first response of contributors

as the first update (i.e., `committed` and `head_ref_force_pushed`), feedback (i.e., `commented`, `reviewed`, `line-commented`, and `commit-commented`, or resolution (i.e., `closed` and `reopened` by the contributor after receiving the first response by a maintainer. We identified bots by marking actors listed in any of the three ground-truth datasets [131, 132, 133], as well as those with names ending in `bot` or `[bot]`. Additionally, we manually inspected actors with high activity levels or fast response times to detect any potential bots that may have been overlooked.

5.2.3 Feature Extraction

To train machine learning models for predicting the first response latency of either maintainers or contributors, we need to extract a set of relevant features that can potentially be predictive of their first response latencies. As outlined in Table 5.2, we extract a total of 21 features covering four different dimensions: (i) project characteristics, (ii) contributor characteristics, (iii) PR characteristics, and (iv) review process characteristics. Features in the table are denoted by 'M', 'C', or 'MC', indicating their use in the models for predicting the response latency of maintainers, contributors, or both, respectively. The features are measured using the data available at different time points: features for predicting the maintainer response latency are measured at the submission time of PR, while features for predicting the contributor response latency are measured at the time when the PR receives its first response from a maintainer. In the following, we explain the relevance and measurement of each feature in detail.

Submission Volume. Increased PR submissions can overwhelm maintainers, resulting in delayed response times [9]. However, a high volume of submissions indicates an active project, which can attract contributors and positively influence their responsiveness. To quantify this feature, we count the number of PRs submitted to the project over the last three months.

Project Backlog. A sizeable backlog of unresolved PRs could overwhelm maintainers, potentially resulting in extended response times [21, 22]. Furthermore, contributors may perceive a substantial backlog as a sign of inattentiveness, which may discourage them and adversely affect their willingness to respond promptly. To quantify this feature, we count the number of unresolved PRs in the project.

Maintainers Availability. More available maintainers facilitate efficient workload distribution, potentially leading to quicker response times [22, 9]. Higher availability of maintainers indicates an actively maintained and supportive project, boosting contributors' confidence and responsiveness. To quantify this feature, we count the number of active maintainers in the project over the last three months.

Table 5.2: Features extracted to predict the first response latency of maintainers (M) and contributors (C).

Dimension	Feature	Description	Model
Project	Submission Volume	Number of submitted PRs to the project over the last 3 months	MC
	Project Backlog	Number of unresolved PRs in the project	MC
	Maintainers Availability	Number of active maintainers in the project over the last 3 months	MC
	Maintainers Responsiveness	Median first response latency of the maintainers over the last 3 months	MC
	Community Size	Number of active community members in the project over the last 3 months	MC
Contributor	Contributor Experience	Number of prior PRs by the contributor	MC
	Contributor Performance	Ratio of the merged PRs of the contributor	MC
	Contributor Backlog	Number of unresolved PRs by the contributor	MC
	Contributor Responsiveness	Median first response latency of the contributor in prior PRs	MC
Pull Request	Description Length	Number of words in the title and description of the PR	MC
	Commits	Number of commits in the PR	MC
	Changed Lines	Number of changed lines in the PR	MC
	Changed Files	Number of changed files in the PR	MC
	Submission Day	Weekday of the submission time of the PR	M
	Submission Hour	Hour of the submission time of the PR	M
Review Process	Review Day	Weekday of the first response of the maintainer	C
	Review Hour	Hour of the first response of the maintainer	C
	Review Latency	First response latency of the maintainer in the PR	C
	Contributor Activity	Number of events by the contributor in the PR	C
	Participants Activity	Number of events by the participants in the PR	C
	Bots Activity	Number of events by the bots in the PR	C

Maintainers Responsiveness. The past responsiveness of maintainers can serve as an indicator of their future response times. Specifically, a history of delayed responses may be interpreted as inattentiveness or lack of engagement, which can demotivate contributors, subsequently dampening their responsiveness [24, 8]. To quantify this feature, we calculate the median first response latency of maintainers for PRs they responded to over the last three months.

Community Size. High levels of community engagement can introduce diverse inputs and increased demands on maintainers, potentially leading to delayed responses [22, 9]. Nevertheless, active community participation reflects a thriving project, which can encourage prompt responses by contributors. To quantify this feature, we count the number of active practitioners (excluding maintainers and bots) in the project over the last three months.

Contributor Experience. Experienced contributors tend to submit higher-quality PRs and communicate more effectively, potentially expediting responses from maintainers [21, 22]. Their familiarity with the project's dynamics and expectations often leads to quicker responses to feedback and revision requests [26]. To quantify this feature, we calculate the number of PRs a contributor has previously submitted to the project.

Contributor Performance. Contributors who consistently have a high success rate with their submissions often produce PRs that align closely with the project's standards, likely receiving quicker responses from

maintainers [21, 22]. On the other hand, contributors who are familiar with the project's expectations typically respond more promptly to feedback [26]. To quantify this feature, we calculate the ratio of a contributor's successfully merged PRs to their total submissions in the project.

Contributor Backlog. A backlog of PRs from a contributor may suggest they are overextended or tend to submit PRs that require extensive review, potentially leading to delayed responses from maintainers [24]. Additionally, having multiple pending submissions may divide the contributor's attention, slowing their response times. To quantify this feature, we count the number of a contributor's unresolved PRs in the project.

Contributor Responsiveness. The past responsiveness of a contributor can serve as an indicator of their future response times. Previous timely responses not only exhibit commitment but also foster a cycle of timely feedback from maintainers [24, 8]. To quantify this feature, we calculate the median latency of the contributor's first responses in prior PRs within the project.

Description Length. A description that is both detailed and concise can significantly aid maintainers in evaluating the proposed changes, potentially resulting in faster response times [21, 22]. A clear description may also minimize the need for further clarification or modifications, allowing contributors to promptly and efficiently address feedback from maintainers [26]. To quantify this feature, we count the number of words in both the title and description of the PR.

Commits. PRs with fewer commits often facilitate a smoother review process, leading to quicker responses from maintainers [21, 22]. On the other hand, contributors who craft meaningful commits are also likely more motivated and attentive to feedback, resulting in prompt responses [26]. To quantify this feature, we count the number of commits submitted to the PR.

Changed Lines. PRs with extensive changes are often more difficult and time-consuming for maintainers to review, leading to delayed responses [21, 22]. Contributors of such PRs also typically need more time to address the changes requested by the maintainers [26]. To quantify this feature, we count the number of lines that have been changed in the commits included in the PR.

Changed Files. PRs that touch multiple files require the reviewer to go through changes across different files, which could extend the time it takes for maintainers to respond [21, 22]. Similar to PRs with many changed lines, those that change many files often reflect a significant effort by contributors, making them more eager to respond to feedback. However, addressing feedback on multiple files may take contributors more time [26]. To quantify this feature, we count the number of files changed in the PR's commits.

Submission Day. PRs submitted closer to weekends tend to receive slower responses due to reduced

activity from maintainers compared to normal working days [134, 9]. To quantify this feature, we record the day of the week on which the PR is submitted, with all times standardized to the UTC timezone.

Submission Hour. Similar to submission day, PRs submitted during the usual working or active hours of maintainers tend to receive quicker responses. In contrast, those submitted outside of these hours might face delays, as maintainers may not be readily available or active [9]. To quantify this feature, we record the hour on which the PR is submitted, with all times standardized to the UTC timezone.

Review Day. Similar to PR submission day, reviews conducted on weekdays might align more closely with the schedules of contributors who approach their work on the project professionally, encouraging timely responses [9]. To quantify this feature, we record the day of the week when the maintainer's first response is submitted, with all times standardized to the UTC timezone.

Review Hour. If the review hour aligns with contributors' usual active hours, it improves the chances that the contributor respond more promptly [9]. To quantify this feature, we record the hour of the maintainer's first response to the PR, with all times standardized to the UTC timezone.

Review Latency. Prior studies show that the first response latency of maintainers is directly correlated with the total duration of the PR review process [21, 22]. Timely responses from maintainers enhance contributors' engagement with the project [26]. However, delays in reviews can lead to frustration and potential PR abandonment [24, 8, 107]. To quantify this feature, we measure the time it takes for maintainers to issue their first response to a PR in hours.

Contributor Activity. The level of activity exhibited by a contributor during the review process can be a sign of their engagement and dedication to the PR, which often correlates with quicker responsiveness to feedback [8]. To quantify this feature, we count the number of events initiated by the contributor following the PR's submission.

Participants Activity. The level of activity from participants, excluding maintainers and bots, during the review process indicates the community's interest and engagement with a specific PR. When a PR attracts attention and constructive comments from the community, it tends to encourage and motivate contributors to respond more quickly [8]. To quantify this feature, we count the number of events initiated by these participants following the PR's submission.

Bots Activity. Bots play an essential role in facilitating the review process by automating repetitive tasks. However, excessive or inappropriate bot activity disrupts contributors, potentially impeding their responsiveness and engagement [29, 34, 28]. To quantify this feature, we count the number of events triggered

by bots after the submission of the PR.

5.2.4 Data Preprocessing

In the following, we explain how we preprocess the dataset before constructing our models.

Data Filtering. We exclude PRs submitted by bots to avoid skewing response latency data. Bots typically receive different treatment from maintainers, leading to atypical response patterns. This can misrepresent the typical response dynamics between human maintainers and contributors as bot interactions do not mirror human behavior.

Feature Correlation Analysis. The presence of multicollinearity [82] can adversely affect both the performance and interpretability of machine learning models [135]. To mitigate this issue, we identify highly correlated features using the Spearman rank correlation test [83] as a nonparametric test that does not require normally distributed data. Then, we conduct the analysis on the entire dataset to ensure that the identified correlations are consistent across all the studied projects. For strongly correlated features with $|\rho| \geq 0.6$ (as suggested by [85]), we keep the most interpretable feature in the context of our study and discard the rest. Accordingly, we dropped *Changed Lines*, *Changed Files*, *Contributor Experience*, and *Submission Volume* for both the maintainers and contributors models.

Feature Transformation. Skewed data can negatively impact machine learning models that expect normally distributed data. To address this issue, we apply log transformation to the features using $\log(x + 1)$. For models other than tree-based ones, where having a comparable scale for features is crucial for optimal performance, we further standardize the features using the Z-score normalization technique to achieve a distribution with a mean of zero and a standard deviation of one.

5.2.5 Model Construction and Evaluation

We build and validate various machine learning models that use the extracted features for predicting the first response latency of maintainers and contributors. The models classify the response times into one of the following three classes: 1) within 1 day, 2) 1 day to 1 week, 3) more than 1 week. This classification scheme is adapted from the study conducted by Hasan et al. [21], which categorized the response times into four groups: 1) within 1 day, 2) 1 day to 1 week, 3) 1 week to 1 month, and 4) more than 1 month. However, we combined the last two categories since PRs with first responses more than one month are rare.

We experiment with various classifier models to identify which ones most accurately predict the first response latency of maintainers and contributors across the studied projects. The selected models include CatBoost (CB) [136], K-Nearest Neighbors (KNN), Logistic Regression (LR), Naive Bayes (NB), Neural Network (NN), Random Forest (RF), and Support Vector Machine (SVM). CatBoost is recommended for multiclass imbalanced datasets [137]. Other models are also popular in the software engineering literature [138, 139]. For each project, we train two distinct models using the corresponding selected features: one focusing on the first response latency of maintainers and another on the first response latency of contributors. To evaluate the performance of our models, we use Time Series cross-validation. This technique is a variation of K-Fold cross-validation that takes into account the temporal nature of our data, thus preventing future information from leaking to training sets. In each split, the first k folds serve as the training set and the $(k+1)$-th fold serves as the test set. To measure the performance of the models, we rely on the following two threshold-independent metrics that are commonly used to evaluate model performance in the presence of an imbalanced dataset [90]:

- **AUC-ROC:** This metric measures the area under the Receiver Operating Characteristic (ROC) curve [91], which plots the true positive rate against the false positive rate at different thresholds. AUC-ROC values range from 0 to 1, with a value greater than 0.5 indicating that the model performs better than a no-skill classifier.
- **AUC-PR:** This metric measures the area under the Precision-Recall (PR) curve [92], which plots the precision against the recall across different thresholds. AUC-PR also ranges from 0 to 1. However, the performance of a no-skill classifier is determined by the class distribution.

These metrics provide a comprehensive assessment of model discriminative capability across various probability thresholds, allowing for a robust and generalized evaluation of performance. To measure these metrics, we adopt the One-vs-Rest (OvR) approach [140] commonly used for evaluating multiclass classifiers [141]. This approach decomposes the multiclass classification problem into multiple binary classification problems, each focusing on a single class against all others. The metrics are then computed for each binary problem and averaged to provide an aggregated overview of the model's capability across different classes. Using this approach, we can perform a detailed evaluation of the model's capability to distinguish each class, ensuring that the model is not biased towards any particular class and can perform well across all classes. Finally, we calculate the relative improvement compared to a no-skill classifier as our baseline. To identify

the best-performing models, we rely on the nonparametric Scott-Knott ESD test [142]. This is a multiple comparison approach that leverages hierarchical clustering to partition the set of median values of techniques into statistically distinct groups with non-negligible differences.

5.2.6 Model Analysis

To compare the relative importance of different features, we perform permutation feature importance analysis [94, 143] for each project. This approach permutes a feature to break the association between the feature and the outcome (i.e., the response latency). The importance of the feature is then measured by how much error the permutated data introduces compared to the original error (i.e., loss in AUC in our case). Therefore, the most important features have the largest impact on the performance of our models and thus are more useful for making accurate predictions. After calculating the importance of each feature in each project, we apply the nonparametric Scott-Knott ESD test [142] to obtain the ranking of each feature. To understand the impact of each feature on the model's predictions, we employ SHapley Additive exPlanation analysis (SHAP) [127, 143]. We calculate the SHAP values for each project and then aggregate the results to gain a holistic understanding of the impact of a feature across all the studied projects.

5.3 Maintainer Response Latency

In this section, we aim to understand if we can accurately predict the first response latency of maintainers following the submission of a PR. Then, we want to identify and discuss the impact of the most important features in accurately predicting the first response latency of maintainers across the studied projects.

5.3.1 RQ1: How well can we predict the first response latency of maintainers?

Table 5.3 and Table 5.4 compare the performance of various models for predicting the first response latency of maintainers in terms of the AUC-ROC and AUC-PR metrics, respectively. The best-performing models according to the Scott-Knott ESD test are highlighted in bold. It is worth noting that multiple models may be identified as best-performing when the difference in their performance is not statistically significant. From the tables, we find that the CB model consistently outperforms all other models in every project in both AUC-ROC and AUC-PR. Compared to the baseline (i.e., a no-skill classifier), the CB model achieves considerable improvements, with increases ranging from 16% to 45% in AUC-ROC and from 20% to 118%

Table 5.3: AUC-ROC of different models for predicting the first response latency of maintainers across the studied projects.

Project	CB	KNN	LR	NB	NN	RF	SVM
Odoo	**0.65 (31%)**	0.59 (18%)	0.63 (26%)	0.62 (24%)	0.64 (28%)	0.60 (20%)	0.63 (26%)
Kubernetes	**0.63 (25%)**	0.56 (12%)	**0.62 (24%)**	0.60 (19%)	0.61 (22%)	0.56 (12%)	0.58 (16%)
Elasticsearch	**0.67 (33%)**	0.58 (15%)	0.63 (25%)	0.61 (22%)	0.64 (29%)	0.56 (12%)	0.57 (15%)
PyTorch	**0.64 (29%)**	0.57 (14%)	0.61 (23%)	0.62 (24%)	0.62 (25%)	0.57 (14%)	0.59 (18%)
Rust	**0.59 (19%)**	0.53 (6%)	0.59 (18%)	0.58 (15%)	0.57 (15%)	0.52 (4%)	0.53 (7%)
DefinitelyTyped	**0.58 (16%)**	0.54 (9%)	**0.58 (16%)**	0.55 (10%)	0.56 (11%)	0.55 (10%)	0.57 (13%)
HomeAssistant	**0.69 (38%)**	0.59 (19%)	0.63 (26%)	0.61 (22%)	0.67 (34%)	0.57 (15%)	0.59 (18%)
Ansible	**0.61 (23%)**	0.56 (12%)	0.60 (20%)	0.57 (15%)	0.60 (20%)	0.57 (14%)	0.60 (20%)
CockroachDB	**0.67 (35%)**	0.58 (15%)	0.66 (32%)	0.63 (25%)	0.64 (27%)	0.58 (15%)	0.57 (14%)
Swift	**0.63 (27%)**	0.55 (10%)	0.61 (21%)	0.59 (19%)	0.62 (23%)	0.54 (7%)	0.55 (9%)
Flutter	**0.71 (42%)**	0.61 (23%)	0.67 (35%)	0.64 (29%)	0.67 (35%)	0.61 (22%)	0.64 (28%)
Spark	**0.65 (30%)**	0.57 (13%)	0.61 (22%)	0.59 (18%)	0.62 (24%)	0.56 (12%)	0.57 (14%)
Python	**0.62 (24%)**	0.55 (10%)	0.60 (20%)	0.57 (15%)	0.60 (19%)	0.56 (12%)	0.59 (18%)
Sentry	**0.72 (45%)**	0.60 (19%)	0.66 (31%)	0.68 (36%)	0.69 (38%)	0.57 (14%)	0.61 (21%)
PaddlePaddle	**0.62 (24%)**	0.56 (11%)	0.59 (18%)	0.60 (19%)	0.60 (21%)	0.55 (11%)	0.58 (15%)
Godot	**0.62 (24%)**	0.55 (11%)	**0.62 (24%)**	0.60 (20%)	0.60 (21%)	0.54 (9%)	0.57 (13%)
Rails	**0.64 (28%)**	0.57 (13%)	0.64 (27%)	0.60 (20%)	0.63 (25%)	0.53 (6%)	0.57 (15%)
Grafana	**0.69 (38%)**	0.59 (19%)	0.65 (31%)	0.65 (31%)	0.67 (35%)	0.58 (15%)	0.60 (20%)
ClickHouse	**0.63 (25%)**	0.55 (10%)	0.57 (14%)	0.58 (15%)	0.60 (20%)	0.56 (11%)	0.57 (14%)
Symfony	**0.62 (24%)**	0.54 (8%)	**0.62 (24%)**	0.60 (20%)	0.60 (19%)	0.52 (3%)	0.54 (9%)
Average	**0.64 (29%)**	0.57 (13%)	0.62 (24%)	0.60 (21%)	0.62 (25%)	0.56 (12%)	0.58 (16%)

Values in parentheses show the percentage improvement compared to the baseline.

in AUC-PR across different projects.

To understand the factors contributing to the misclassification of the first response latency of maintainers, we manually examined the misclassified PRs by the CB model. The most important reason is that the predictions are based on the data available up until the time of PR submission. However, there are various post-submission factors that affect how long it takes for maintainers to respond. For example, if a PR fails Continuous Integration (CI) tests, maintainers usually wait for the errors to be fixed before starting the review [24, 8]. Furthermore, the presence of bot-generated responses is known to prolong the first response latency of maintainers [21].

The CB model can predict the first response latency of maintainers with an average improvement of 29% in AUC-ROC and 51% in AUC-PR compared to a no-skilled classifier.

Table 5.4: AUC-PR of different models for predicting the first response latency of maintainers across the studied projects.

Project	CB	KNN	LR	NB	NN	RF	SVM
Odoo	**0.48 (45%)**	0.40 (21%)	0.44 (34%)	0.43 (30%)	0.46 (39%)	0.40 (19%)	0.45 (34%)
Kubernetes	**0.42 (34%)**	0.37 (11%)	0.42 (35%)	0.40 (25%)	0.41 (28%)	0.36 (10%)	0.39 (21%)
Elasticsearch	**0.44 (58%)**	0.37 (17%)	0.41 (44%)	0.39 (37%)	0.42 (49%)	0.37 (15%)	0.38 (25%)
PyTorch	**0.44 (43%)**	0.37 (15%)	0.41 (32%)	0.41 (32%)	0.42 (36%)	0.37 (13%)	0.40 (28%)
Rust	**0.38 (30%)**	0.35 (6%)	0.37 (25%)	0.37 (21%)	0.37 (21%)	0.34 (4%)	0.35 (12%)
DefinitelyTyped	**0.41 (20%)**	0.36 (9%)	**0.40 (21%)**	0.37 (10%)	0.39 (15%)	0.36 (8%)	0.39 (16%)
HomeAssistant	**0.42 (79%)**	0.37 (25%)	0.38 (45%)	0.38 (35%)	0.41 (67%)	0.36 (22%)	0.37 (39%)
Ansible	**0.43 (33%)**	0.38 (13%)	0.41 (26%)	0.39 (17%)	0.42 (27%)	0.37 (12%)	0.42 (27%)
CockroachDB	**0.42 (67%)**	0.37 (19%)	0.42 (59%)	0.39 (44%)	0.41 (48%)	0.37 (19%)	0.38 (35%)
Swift	**0.40 (50%)**	0.35 (13%)	0.38 (36%)	0.37 (24%)	0.38 (41%)	0.35 (9%)	0.35 (20%)
Flutter	**0.47 (76%)**	0.40 (32%)	0.44 (60%)	0.41 (47%)	0.45 (63%)	0.39 (29%)	0.43 (52%)
Spark	**0.42 (52%)**	0.36 (17%)	0.39 (31%)	0.38 (23%)	0.40 (39%)	0.36 (13%)	0.37 (24%)
Python	**0.41 (30%)**	0.36 (10%)	0.40 (26%)	0.38 (19%)	0.40 (24%)	0.37 (12%)	0.40 (23%)
Sentry	**0.45 (118%)**	0.37 (31%)	0.41 (82%)	0.40 (71%)	0.42 (105%)	0.37 (26%)	0.38 (57%)
PaddlePaddle	**0.42 (37%)**	0.37 (15%)	0.39 (29%)	0.39 (28%)	0.40 (31%)	0.36 (10%)	0.39 (25%)
Godot	**0.41 (35%)**	0.36 (12%)	0.40 (31%)	0.39 (24%)	0.40 (29%)	0.35 (8%)	0.37 (18%)
Rails	**0.40 (43%)**	0.36 (13%)	0.40 (42%)	0.38 (25%)	0.39 (34%)	0.34 (8%)	0.36 (21%)
Grafana	**0.45 (82%)**	0.38 (26%)	0.42 (67%)	0.41 (57%)	0.43 (72%)	0.37 (20%)	0.39 (44%)
ClickHouse	**0.41 (43%)**	0.36 (12%)	0.38 (23%)	0.38 (26%)	0.40 (35%)	0.36 (11%)	0.38 (27%)
Symfony	**0.39 (39%)**	0.35 (10%)	0.38 (38%)	0.38 (27%)	0.37 (28%)	0.34 (5%)	0.35 (19%)
Average	**0.42 (51%)**	0.37 (16%)	0.40 (39%)	0.39 (31%)	0.41 (41%)	0.36 (14%)	0.38 (28%)

Values in parentheses show the percentage improvement compared to the baseline.

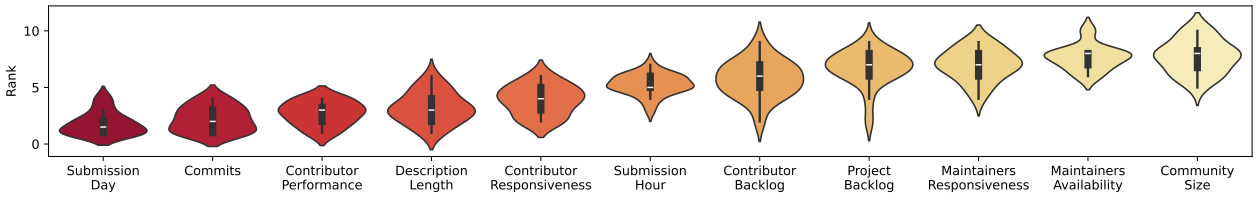


Figure 5.1: Ranking of the importance of different features for predicting the first response latency of maintainers across the studied projects. Darker colors indicate higher importance.

5.3.2 RQ2: What are the major predictors of the first response latency of maintainers?

To conduct this analysis, we use the CB models due to their superior performance. Figure 5.1 overviews the rankings of different features based on their importance in accurately predicting the first response latency of maintainers across the studied projects. From the figure, we observe that *Submission Day*, *Commits*, *Contributor Performance*, *Description Length*, and *Contributor Responsiveness* are the top five most important features. This observation implies that the characteristics of PRs and contributors have a greater influence on how quickly maintainers provide their first response compared to project characteristics. We were surprised by this observation, as we expected that at least historical maintainer responsiveness to be a key predictor of future response times.

Figure 5.2 illustrates the impact of the top five most important features on the probability of receiving the first maintainer response on the same day of submitting a PR. We observe that PRs submitted earlier in the week, containing an average or slightly above-average number of commits at submission, and with more concise descriptions are more likely to get faster responses. Similarly, contributors with a higher acceptance rate and a history of timely responses in the project tend to obtain quicker responses. However, it is concerning that inexperienced contributors are prone to encounter delays in receiving feedback. This lack of timely responsiveness from maintainers is frequently cited as a key reason why contributors, especially novice or casual contributors, may abandon their PRs [24, 8, 107] and even cease further contributing to the project [25, 104].

The *Submission Day*, *Commits*, *Contributor Performance*, *Description Length*, and *Contributor Responsiveness* have the most influence on the first response latency of maintainers.

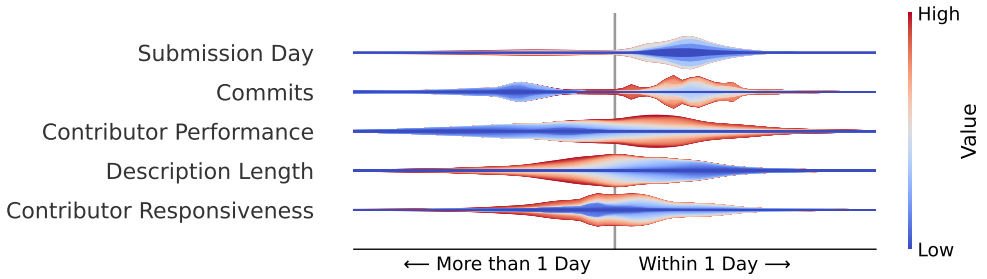


Figure 5.2: Impact of the top 5 most important features on the prediction of the first response latency of maintainers across the studied projects. Wider violins indicate higher density and more frequent values.

5.4 Contributor Response Latency

In this section, we aim to understand if we can accurately predict the first response latency of the contributor of a PR after receiving the first response from the maintainers. Then, we want to identify and discuss the impact of the most important features in accurately predicting the first response latency of contributors across the studied projects.

5.4.1 RQ3: How well can we predict the first response latency of contributors?

Table 5.5 and Table 5.6 compare the performance of various models for predicting the first response latency of contributors in terms of the AUC-ROC and AUC-PR metrics, respectively. The best-performing models according to the Scott-Knott ESD test are highlighted in bold. It is worth noting that multiple models may be identified as best-performing when the difference in their performance is not statistically significant. From the tables, we find that similar to the first response latency of maintainers (see Section 5.3), the CB model continues to demonstrate superior performance in both AUC-ROC and AUC-PR across all projects. Compared to the baseline (i.e., a no-skill classifier), the CB model achieves significant improvements, with increases ranging from 24% to 50% in AUC-ROC and from 39% to 142% in AUC-PR across different projects.

To understand the factors contributing to the misclassification of the first response latency of contributors, we manually examined the misclassified PRs by the CB model. We find that the quality of feedback and the extent of requested changes also influence how long it takes for contributors to respond after the feedback. This observation aligns with prior findings in the literature that emphasize the importance of quality review comments from the maintainers [5, 144].

Table 5.5: AUC-ROC of different models for predicting the first response latency of contributors across the studied projects.

Project	CB	KNN	LR	NB	NN	RF	SVM
Odoo	**0.66 (32%)**	0.55 (11%)	0.64 (28%)	0.62 (24%)	0.62 (24%)	0.57 (13%)	0.59 (19%)
Kubernetes	**0.68 (37%)**	0.58 (16%)	0.68 (35%)	0.64 (27%)	0.65 (30%)	0.59 (17%)	0.61 (21%)
Elasticsearch	**0.74 (48%)**	0.59 (18%)	0.71 (41%)	0.67 (35%)	0.69 (38%)	0.58 (16%)	0.60 (21%)
PyTorch	**0.68 (35%)**	0.58 (15%)	**0.67 (33%)**	0.64 (29%)	0.64 (28%)	0.58 (16%)	0.60 (20%)
Rust	**0.64 (28%)**	0.56 (12%)	**0.64 (28%)**	0.62 (23%)	0.61 (21%)	0.56 (12%)	0.55 (11%)
DefinitelyTyped	**0.62 (24%)**	0.55 (10%)	0.61 (21%)	0.59 (18%)	0.60 (21%)	0.54 (8%)	0.56 (11%)
HomeAssistant	**0.75 (49%)**	0.62 (23%)	0.72 (44%)	0.68 (36%)	0.71 (43%)	0.58 (15%)	0.58 (17%)
Ansible	**0.66 (32%)**	0.58 (17%)	**0.66 (33%)**	0.61 (22%)	0.64 (27%)	0.57 (15%)	0.61 (22%)
CockroachDB	**0.72 (44%)**	0.58 (15%)	0.71 (41%)	0.66 (32%)	0.66 (31%)	0.57 (14%)	0.58 (16%)
Swift	**0.72 (44%)**	0.58 (16%)	0.69 (38%)	0.68 (35%)	0.67 (34%)	0.56 (12%)	0.59 (19%)
Flutter	**0.72 (44%)**	0.59 (17%)	0.69 (37%)	0.62 (23%)	0.65 (30%)	0.56 (13%)	0.60 (20%)
Spark	**0.73 (45%)**	0.59 (18%)	0.70 (40%)	0.69 (38%)	0.68 (36%)	0.57 (14%)	0.61 (21%)
Python	**0.68 (35%)**	0.58 (15%)	**0.67 (34%)**	0.66 (31%)	0.65 (31%)	0.54 (9%)	0.58 (16%)
Sentry	**0.74 (49%)**	0.58 (17%)	0.69 (38%)	0.66 (33%)	0.69 (38%)	0.58 (16%)	0.60 (20%)
PaddlePaddle	**0.67 (35%)**	0.55 (10%)	0.64 (28%)	0.62 (24%)	0.61 (22%)	0.55 (10%)	0.56 (12%)
Godot	**0.67 (35%)**	0.59 (18%)	**0.68 (35%)**	0.66 (32%)	0.66 (31%)	0.56 (12%)	0.61 (23%)
Rails	**0.68 (36%)**	0.57 (13%)	0.66 (33%)	0.62 (25%)	0.62 (24%)	0.55 (10%)	0.58 (15%)
Grafana	**0.75 (50%)**	0.60 (21%)	0.71 (42%)	0.70 (40%)	0.71 (41%)	0.59 (17%)	0.60 (20%)
ClickHouse	**0.70 (41%)**	0.57 (14%)	0.69 (38%)	0.64 (28%)	0.67 (34%)	0.57 (13%)	0.59 (19%)
Symfony	**0.68 (36%)**	0.57 (15%)	**0.68 (35%)**	0.66 (31%)	0.64 (29%)	0.55 (11%)	0.59 (18%)
Average	**0.69 (39%)**	0.58 (16%)	0.68 (35%)	0.65 (29%)	0.65 (31%)	0.57 (13%)	0.59 (18%)

Values in parentheses show the percentage improvement compared to the baseline.

Table 5.6: AUC-PR of different models for predicting the first response latency of contributors across the studied projects.

Project	**CB**	**KNN**	**LR**	**NB**	**NN**	**RF**	**SVM**
Odoo	**0.44 (58%)**	0.36 (12%)	0.42 (49%)	0.40 (35%)	0.41 (39%)	0.37 (15%)	0.39 (34%)
Kubernetes	**0.45 (62%)**	0.37 (19%)	0.44 (60%)	0.41 (40%)	0.43 (48%)	0.38 (20%)	0.40 (37%)
Elasticsearch	**0.46 (142%)**	0.37 (28%)	0.43 (112%)	0.40 (72%)	0.42 (97%)	0.37 (32%)	0.38 (58%)
PyTorch	**0.45 (70%)**	0.37 (19%)	0.43 (60%)	0.41 (45%)	0.42 (55%)	0.37 (20%)	0.40 (39%)
Rust	**0.41 (46%)**	0.36 (14%)	**0.41 (51%)**	0.39 (35%)	0.39 (36%)	0.36 (13%)	0.36 (22%)
DefinitelyTyped	**0.40 (39%)**	0.36 (13%)	0.39 (30%)	0.38 (23%)	0.39 (34%)	0.35 (9%)	0.37 (20%)
HomeAssistant	**0.44 (112%)**	0.37 (36%)	0.43 (103%)	0.41 (73%)	0.42 (93%)	0.37 (29%)	0.38 (48%)
Ansible	**0.45 (51%)**	0.38 (21%)	**0.45 (50%)**	0.41 (32%)	0.42 (40%)	0.37 (16%)	0.41 (34%)
CockroachDB	**0.43 (95%)**	0.36 (21%)	0.42 (95%)	0.39 (52%)	0.39 (61%)	0.36 (25%)	0.37 (38%)
Swift	**0.43 (108%)**	0.36 (24%)	0.41 (99%)	0.39 (69%)	0.40 (72%)	0.36 (31%)	0.37 (54%)
Flutter	**0.44 (141%)**	0.37 (30%)	0.42 (125%)	0.38 (50%)	0.40 (68%)	0.36 (42%)	0.38 (50%)
Spark	**0.45 (106%)**	0.37 (27%)	0.44 (104%)	0.42 (75%)	0.42 (77%)	0.37 (24%)	0.39 (59%)
Python	**0.42 (63%)**	0.36 (20%)	**0.42 (65%)**	0.40 (51%)	0.41 (56%)	0.35 (12%)	0.38 (37%)
Sentry	**0.46 (153%)**	0.36 (21%)	0.42 (121%)	0.38 (64%)	0.41 (87%)	0.38 (45%)	0.37 (48%)
PaddlePaddle	**0.41 (76%)**	0.35 (14%)	0.39 (65%)	0.38 (47%)	0.38 (50%)	0.35 (19%)	0.36 (30%)
Godot	**0.42 (63%)**	0.37 (26%)	**0.42 (66%)**	0.40 (52%)	**0.41 (62%)**	0.36 (20%)	0.39 (46%)
Rails	**0.41 (76%)**	0.36 (20%)	**0.41 (73%)**	0.38 (47%)	0.38 (50%)	0.36 (22%)	0.37 (40%)
Grafana	**0.44 (120%)**	0.37 (30%)	0.43 (114%)	0.40 (79%)	0.41 (91%)	0.37 (29%)	0.37 (51%)
ClickHouse	**0.46 (114%)**	0.37 (34%)	**0.45 (115%)**	0.40 (52%)	0.43 (93%)	0.37 (34%)	0.39 (66%)
Symfony	**0.42 (78%)**	0.36 (24%)	**0.41 (77%)**	0.39 (50%)	0.40 (57%)	0.36 (25%)	0.38 (46%)
Average	**0.43 (89%)**	0.36 (23%)	0.42 (82%)	0.40 (52%)	0.41 (63%)	0.36 (24%)	0.38 (43%)

Values in parentheses show the percentage improvement compared to the baseline.

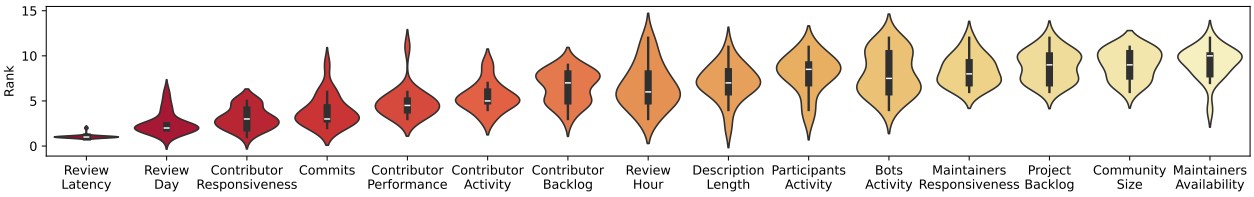


Figure 5.3: Ranking of the importance of different features for predicting the first response latency of contributors across the studied projects. Darker colors indicate higher importance.

The CB model can predict the first response latency of contributors with an average improvement of 39% in AUC-ROC and 89% in AUC-PR compared to a no-skilled classifier.

5.4.2 RQ4: What are the major predictors of the first response latency of contributors?

To conduct this analysis, we use the CB models due to their superior performance. Figure 5.3 overviews the rankings of different features based on their importance in accurately predicting the first response latency of contributors across the studied projects. From the figure, we observe that *Review Latency*, *Review Day*, *Contributor Responsiveness*, *Commits*, and *Contributor Performance* are the top five most important features. This observation highlights the great influence of the characteristics of the review process and contributors on how quickly the contributors first respond.

Figure 5.4 illustrates the the impact of the top five most important features on the probability of contributors replying on the same day they receive the first maintainers response. We observe that lower latency in the first maintainer response correlates with a quicker subsequent response from the contributor. In other words, contributors tend to reply late if they have experienced delayed responses, leading to a cascade effect in the review process. Notably, the first response latency of maintainers not only affects the contributor responsiveness but is also known to directly impact the duration [21, 22] and the outcome [23, 24, 8] of the PR, as well as the likelihood of future contributions by the contributor to the project [25, 21]. Furthermore, we find contributors of PRs that received the first response of maintainers earlier in the week, and contributors of PRs containing an average or slightly above-average number of commits till the first response of maintainers are more likely to provide faster responses. Similarly, contributors with a history of timely responses in the project and with a higher acceptance rate tend to give quicker responses.

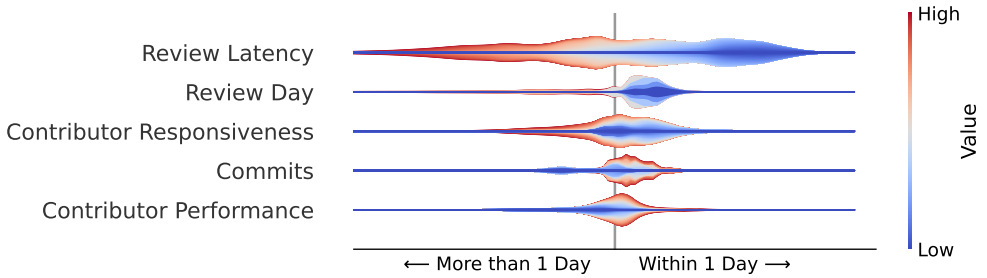


Figure 5.4: Impact of the top 5 most important features on the prediction of the first response latency of contributors across the studied projects. Wider violins indicate higher density and more frequent values.

The *Review Latency*, *Review Day*, *Contributor Responsiveness*, *Commits*, and *Contributor Performance* have the most influence on the first response latency of contributors.

5.5 Cross-Project Setting

Building accurate predictive models for new projects is often challenging due to the limited historical data available. However, one way to overcome this challenge is through cross-project prediction, which enables such projects to leverage the insights and patterns observed in older, well-established projects. To evaluate the effectiveness of this approach, for each studied project, we train a CB model (as it has shown superior performance in our previous analyses) on all other projects and then test on that project.

Table 5.7 compares the performance of various models in predicting the first response latency of maintainers and contributors in a cross-project setting in terms of the AUC-ROC and AUC-PR metrics, respectively. We observe that the models for predicting the maintainers response latency achieve improvements ranging from 22% to 58% in AUC-ROC and from 28% to 122% in AUC-PR. The models for predicting the contributors response latency also demonstrate improvements ranging from 26% to 56% in AUC-ROC and from 35% to 149% in AUC-PR. Furthermore, we find that *Submission Day*, *Commits*, *Contributor Performance*, *Maintainers Responsiveness*, and *Contributor Responsiveness* are the most important for the first response latency of maintainers; and *Review Latency*, *Review Day*, *Contributor Responsiveness*, *Commits*, and *Contributor Activity* are the major predictors of the first response latency of contributors (see Appendix 6.2.5). The results indicate that our approach can be effective for predicting both the first response latency of maintainers and the first response latency of contributors in a cross-project setting.

Table 5.7: Performance of the models for predicting the first response latency of maintainers and contributors in a cross-project setting.

	Maintainers		Contributors	
Project	**AUC-ROC**	**AUC-PR**	**AUC-ROC**	**AUC-PR**
Odoo	0.64 (28%)	0.46 (37%)	0.66 (32%)	0.44 (57%)
Kubernetes	0.68 (36%)	0.45 (50%)	0.70 (40%)	0.46 (73%)
Elasticsearch	0.67 (35%)	0.44 (61%)	0.74 (49%)	0.46 (133%)
PyTorch	0.66 (33%)	0.45 (50%)	0.70 (40%)	0.46 (71%)
Rust	0.64 (29%)	0.40 (48%)	0.67 (35%)	0.43 (65%)
DefinitelyTyped	0.61 (22%)	0.43 (28%)	0.63 (26%)	0.41 (35%)
HomeAssistant	0.65 (30%)	0.39 (55%)	0.74 (48%)	0.44 (106%)
Ansible	0.64 (28%)	0.45 (38%)	0.69 (39%)	0.47 (69%)
CockroachDB	0.74 (49%)	0.45 (100%)	0.74 (48%)	0.44 (121%)
Swift	0.68 (35%)	0.41 (63%)	0.73 (46%)	0.43 (111%)
Flutter	0.79 (58%)	0.50 (122%)	0.78 (56%)	0.46 (149%)
Spark	0.67 (34%)	0.44 (61%)	0.73 (46%)	0.46 (111%)
Python	0.61 (22%)	0.41 (29%)	0.69 (39%)	0.43 (66%)
Sentry	0.71 (42%)	0.44 (107%)	0.74 (47%)	0.44 (140%)
PaddlePaddle	0.65 (31%)	0.43 (43%)	0.67 (33%)	0.40 (72%)
Godot	0.63 (27%)	0.41 (35%)	0.68 (37%)	0.42 (69%)
Rails	0.66 (33%)	0.43 (64%)	0.71 (43%)	0.44 (101%)
Grafana	0.70 (40%)	0.45 (77%)	0.76 (52%)	0.45 (136%)
ClickHouse	0.62 (24%)	0.40 (35%)	0.72 (44%)	0.48 (122%)
Symfony	0.64 (28%)	0.41 (53%)	0.70 (40%)	0.42 (85%)
Average	0.67 (33%)	0.43 (58%)	0.71 (42%)	0.44 (95%)

Values in parentheses show the percentage improvement compared to the baseline.

5.6 Limitations

In this section, we discuss the threats to the validity of our study.

5.6.1 Internal Validity

The first threat relates to the completeness of our bot detection approach. To mitigate this threat, we manually examined actors with high activity levels or fast response times and added them to our bot list accordingly. The second threat relates to the suitability of our features. To mitigate this threat, we consulted the literature on pull-based development [22, 23] and also drew from our previous experience studying PR abandonment [24, 28]. Still, there could be additional features that we did not consider or that are difficult to quantify, such as code quality, feedback quality, and PR urgency. The third threat relates to the choice of classifiers. To mitigate this threat, we employed seven different classifiers commonly used in the software engineering literature. CatBoost is also acclaimed for its effectiveness with multi-class imbalanced datasets, making it particularly suitable for our study.

5.6.2 External Validity

Our study is based on 20 large and popular open-source projects on GitHub. Although we believe these projects are more likely to benefit from our approach due to their higher activity levels, we recognize that they cannot represent the entire open-source ecosystem. In other words, the studied projects may not represent other open-source projects with different sizes, maturity, popularity, workload, dynamics, social structures, and development practices. Still, we evaluated our approach in a cross-project setting to demonstrate its effectiveness across different projects. Future research can replicate our approach using a more diverse selection of projects.

5.7 Chapter Summary

The objective of this chapter was to develop the first machine learning approach for predicting the latency of both maintainers' first response following a PR submission and the contributor's first response after receiving the first maintainer feedback. We curated a dataset of 20 large and popular open-source projects on GitHub and extracted 21 features characterizing projects, contributors, PRs, and review processes. The

analyses demonstrated the effectiveness of our approach in predicting the first response latency of maintainers and contributors both in project-specific and cross-project settings. Furthermore, the findings highlighted the significant influence of the timing of submissions and feedback, the complexity of changes, and the track record of contributors on the response latencies. By providing estimates of waiting times, our approach can help open-source projects facilitate collaboration between their maintainers and contributors by enabling them to anticipate and address potential delays proactively during the review process of PRs.

Chapter 6

Conclusion and Future Work

In this chapter, we conclude the thesis by summarizing the main work and contributions in each chapter of the thesis. At the end of the chapter, we also discuss some directions for future research.

6.1 Conclusion

In this thesis, we investigated the underlying dynamics of abandoned PRs, evaluated the helpfulness of Stale bot as a common solution for dealing with abandoned PRs, and proposed an approach to predict the first response latency of maintainers and contributors in PRs towards helping to better mitigate PR abandonment in large open-source projects. In this section, we summarize the work done in the thesis.

Chapter 3: Understanding the Dynamics of Contributor-Abandoned Pull Requests

To better understand the underlying dynamics of contributor-abandoned PRs, we conducted a mixed-methods study of 10 large open-source projects on GitHub. Using statistical and machine learning techniques, we found that complex PRs, novice contributors, and lengthy reviews have a higher probability of abandonment and the rate of PR abandonment fluctuates alongside the projects' maturity or workload. To identify why contributors abandon their PRs, we also manually examined a random sample of abandoned PRs. We observed that the most frequent abandonment reasons are related to the obstacles faced by contributors, followed by the hurdles imposed by maintainers during the review process. Finally, we surveyed the top core maintainers of the studied projects to understand their perspectives on dealing with PR abandonment and on our findings.

Chapter 4: Understanding the Helpfulness of Stale Bot for Pull-based Development

To better understand if and how Stale bot helps projects in their pull-based development workflow, we conducted a quantitative empirical study of 20 large open-source projects on GitHub. We found that Stale bot can help deal with a backlog of unresolved PRs as the projects closed more PRs within the first few months of adoption. Moreover, Stale bot can help improve the efficiency of the PR review process as the projects reviewed PRs that ended up merged and resolved PRs that ended up closed faster after the adoption. However, Stale bot can also negatively affect the contributors as the projects experienced a considerable decrease in their number of active contributors after the adoption. Therefore, relying solely on Stale bot to deal with inactive PRs may lead to decreased community engagement and an increased probability of contributor abandonment.

Chapter 5: Predicting the First Response Latency of Maintainers and Contributors in Pull Requests

We propose a machine learning approach to predict the first response latency of the maintainers following the submission of a PR, and the first response latency of the contributor after receiving the first response from the maintainers. We evaluated the effectiveness of our approach on 20 large open-source projects on GitHub. We also conducted permutation feature importance and SHAP analyses to understand the importance and impact of different features on the predicted response latencies. We found that our approach is effective in predicting the first response latency of maintainers and contributors both in project-specific and cross-project settings. Our findings also indicated the significant influence of the timing of submissions and feedback, the complexity of changes, and the track record of contributors on the response latencies. By providing estimates of waiting times, our approach can help open-source projects facilitate collaboration between their maintainers and contributors by enabling them to anticipate and address potential delays proactively during the review process of PRs.

6.2 Future Work

In this section, we present an overview of the key directions for future work.

6.2.1 Extension of Our Proposed Approach in Practice

In Chapter 5, we developed machine learning models to predict the first response latency of maintainers and contributors during the review process of PRs. A promising direction for future work is the development of a comprehensive tool based on our proposed approach. This tool would not only predict anticipated waiting times but also explain the specific reasons for these delays, utilizing techniques such as LIME (Local Interpretable Model-agnostic Explanations) [145]. The tool's functionality could also extend beyond diagnostics to intervention, by proactively providing customized recommendations to minimize potential waiting times. This feature would enhance the tool's practicality and relevance in real-world scenarios. A crucial aspect of this future work is the empirical evaluation of the tool. This evaluation should assess the tool's impact on enhancing collaboration and productivity among maintainers and contributors. Key metrics for this assessment can include the decrease in response times, the increase in PR success rates, and the overall satisfaction of all parties involved.

6.2.2 Assistance in Addressing Change Requests in PRs

In Chapter 3, we observed that the majority of contributors in abandoned PRs lacked the necessary knowledge and experience to effectively address the requested changes by maintainers. Recognizing this challenge, an important direction for future work is to investigate the potential of Large Language Models (LLMs) such as GPT-4 to provide specific instructions or generate required changes to address the maintainers' requests. Such an approach could empower contributors to more effectively respond to change requests, potentially reducing both the rate of PR abandonment and the review duration.

6.2.3 Assistance in Resolution of CI Failures in PRs

Our investigation in Chapter 3 uncovered the prevalence of Continuous Integration (CI) failures among abandoned PRs. We observed that contributors frequently struggle with resolving CI failures that arise during the review process and seek assistance from project maintainers. The project maintainers also emphasized the critical role of clear and actionable error messages in mitigating the abandonment of PRs. Building upon these findings, future work should investigate the characteristics and challenges encountered in PRs blocked by CI failures. The goal is to develop supportive solutions that can diagnose the root causes of the failures and provide context-aware recommendations for resolving them. Such solutions could reduce the cognitive

load on both contributors and maintainers, potentially enhancing the success rate and efficiency of the review process of PRs.

6.2.4 Automatic Resolution of Merge Conflicts in PRs

In Chapter 3, another common challenge observed among abandoned PRs is the occurrence of merge conflicts. These conflicts typically arise when the project's codebase has been updated, requiring contributors to rebase their local branches, address any conflicts, and then push their updated changes. Merge conflicts can significantly hinder the review process, leading to delays and extra work for contributors. A critical area for future work is the development of tools and algorithms to facilitate the automatic resolution of merge conflicts in PRs. The focus of this work should be on continuously predicting and resolving potential conflicts before they arise. Such automation could minimize delays and frustrations associated with manual conflict resolution, potentially leading to a more productive review process.

6.2.5 Smooth Handover Mechanism in PRs

Finally, there are instances where contributors are unable to continue the review process as discussed in Chapter 3. Such situations may arise due to a lack of time, loss of interest, or other personal and professional circumstances. Therefore, a key area for future work is the development of a mechanism that enables the smooth handover of an existing PR to another contributor without requiring the submission of a new PR. The goal is to ensure that the authorship credits and the existing review discussions in the PR are preserved and continued seamlessly. This approach could also involve recommending a successor based on relevant experience, current interest in the project, or the potential benefit associated with continuing the PR.

Bibliography

[1] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, pages 345–355, 2014. doi:10.1145/2568225.2568260.

[2] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Effectiveness of code contribution: from patch-based to pull-request-based tools. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*, pages 871–882, 2016. doi:10.1145/2950290.2950364.

[3] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. pages 1277–1286, 2012. doi:10.1145/2145204.2145396.

[4] Jing Jiang, Jiangfeng Lv, Jiateng Zheng, and Li Zhang. How developers modify pull requests in code review. *IEEE Transactions on Reliability*, 71(3):1325–1339, 2022. doi:10.1109/TR.2021.3093159.

[5] Zhixing Li, Yue Yu, Tao Wang, ShanShan Li, and Huaiming Wang. Opportunities and challenges in repeated revisions to pull-requests: an empirical study. *Proceedings of the ACM on Human-Computer Interaction*, 6(CSCW2):1–35, 2022. doi:10.1145/3555208.

[6] Gustavo Pinto, Igor Steinmacher, and Marco Aurlio Gerosa. More common than you think: an in-depth study of casual contributors. In *Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, pages 112–123, 2016. doi:10.1109/SANER.2016.68.

[7] Noah Davis. 8% of pull requests are doomed, 2018. URL https://codeclimate.com/blog/abandoned-pull-requests.

[8] Zhixing Li, Yue Yu, Tao Wang, Gang Yin, ShanShan Li, and Huaimin Wang. Are you still working on this? An empirical study on pull request abandonment. *IEEE Transactions on Software Engineering*, 48(6):2173–2188, 2022. doi:10.1109/TSE.2021.3053403.

[9] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the IEEE/ACM 37th International Conference on Software Engineering (ICSE 2015)*, pages 358–368, 2015. doi:10.1109/ICSE.2015.55.

[10] Poul Kjeldager Srensen. Pull Request #2143 - DefinitelyTyped/DefinitelyTyped, 2014. URL https://github.com/DefinitelyTyped/DefinitelyTyped/pull/2143.

[11] GitHub. Stale - GitHub Marketplace, 2023. URL https://github.com/marketplace/stale.

[12] Mairieli Wessel, Igor Steinmacher, Igor Wiese, and Marco Aurlio Gerosa. Should I stale or should I close? An analysis of a bot that closes abandoned issues and pull requests. In *Proceedings of the IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE 2019)*, pages 38–42, 2019. doi:10.1109/BotSE.2019.00018.

[13] Brandon Keepers. Stale - GitHub Repository, 2023. URL https://github.com/probot/stale.

[14] Drew DeVault. GitHub Stale bot considered harmful, 2021. URL https://drewdevault.com/2021/10/26/stalebot.html.

[15] Ben Winding. GitHub Stale bots: a false economy, 2021. URL https://blog.benwinding.com/github-stale-bots/index.html.

[16] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco Aurlio Gerosa. Don't disturb me: challenges of interacting with software bots on open source software projects. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–21, 2021. doi:10.1145/3476042.

[17] Dongyu Liu, Micah J. Smith, and Kalyan Veeramachaneni. Understanding user-bot interactions for small-scale automation in open-source development. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA 2020)*, pages 1–8, 2020. doi:10.1145/3334480.3382998.

[18] Mairieli Wessel and Igor Steinmacher. The inconvenient side of software bots on pull requests. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW 2020)*, pages 51–55, 2020. doi:10.1145/3387940.3391504.

[19] Juan Carlos Farah, Basile Spaenlehauer, Xinyang Lu, Sandy Ingram, and Denis Gillet. An exploratory study of reactions to bot comments on GitHub. In *Proceedings of the 4th International Workshop on Bots in Software Engineering (BotSE 2022)*, pages 18–22, 2022. doi:10.1145/3528228.3528409.

[20] Akond Rahman, Farzana Ahamed Bhuiyan, Mohammad Mehedi Hassan, Hossain Shahriar, and Fan Wu. Towards automation for MLOps: an exploratory study of bot usage in deep learning libraries. In *Proceedings of the IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC 2022)*, pages 1093–1097, 2022. doi:10.1109/COMPSAC54236.2022.00171.

[21] Kazi Amit Hasan, Marcos Macedo, Yuan Tian, Bram Adams, and Steven Ding. Understanding the time to first response in GitHub pull requests. In *Proceedings of the IEEE/ACM 20th International Conference on Mining Software Repositories (MSR 2023)*, pages 1–11, 2023. doi:10.1109/MSR59073.2023.00015.

[22] Xunhui Zhang, Yue Yu, Tao Wang, Ayushi Rastogi, and Huaimin Wang. Pull request latency explained: an empirical overview. *Empirical Software Engineering*, 27(6):1–38, 2022. doi:10.1007/s10664-022-10143-4.

[23] Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. Pull request decisions explained: an empirical overview. *IEEE Transactions on Software Engineering*, 49(2):849–871, 2023. doi:10.1109/TSE.2022.3165056.

[24] SayedHassan Khatoonabadi, Diego Elias Costa, Rabe Abdalkareem, and Emad 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*, 32(1):1–39, 2023. doi:10.1145/3530785.

[25] Noppadol Assavakamhaenghan, Supatsara Wattanakriengkrai, Naomichi Shimada, Raula Gaikovina Kula, Takashi Ishio, and Kenichi Matsumoto. Does the first response matter for future contributions? A study of first contributions. *Empirical Software Engineering*, 28(3):1–22, 2023. doi:10.1007/s10664-023-10299-7.

[26] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the IEEE/ACM 38th International Conference on Software Engineering (ICSE 2016)*, pages 285–296, 2016. doi:10.1145/2884781.2884826.

[27] Anita K. Wagner, Stephen B. Soumerai, Fang Zhang, and Dennis Ross-Degnan. Segmented regression analysis of interrupted time series studies in medication use research. *Journal of Clinical Pharmacy and Therapeutics*, 27(4):299–309, 2002. doi:10.1046/j.1365-2710.2002.00430.x.

[28] SayedHassan Khatoonabadi, Diego Elias Costa, Suhaib Mujahid, and Emad Shihab. Understanding the helpfulness of Stale bot for pull-based development: an empirical study of 20 large open-source projects. *ACM Transactions on Software Engineering and Methodology*, 2023. doi:10.1145/3624739.

[29] Mairieli Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco Aurlio Gerosa, and Igor Steinmacher. Bots for pull requests: the good, the bad, and the promising. In *Proceedings of the 44th International Conference on Software Engineering (ICSE 2022)*, pages 274–286, 2022. doi:10.1145/3510003.3512765.

[30] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco Aurlio Gerosa. The power of bots: characterizing and understanding bots in OSS projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–19, 2018. doi:10.1145/3274451.

[31] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*, pages 928–931, 2016. doi:10.1145/2950290.2983989.

[32] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*, pages 60–71, 2017. doi:10.1109/ASE.2017.8115619.

[33] Nathan Cassee, Bogdan Vasilescu, and Alexander Serebrenik. The silent helper: the impact of continuous integration on code reviews. In *Proceedings of the 27th International Confer-*

ence on Software Analysis, Evolution, and Reengineering (SANER 2020), pages 423–434, 2020. doi:10.1109/SANER48275.2020.9054818.

[34] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco Aurlio Gerosa. Quality gatekeepers: investigating the effects of code review bots on pull request activities. *Empirical Software Engineering*, 27(5):1–36, 2022. doi:10.1007/s10664-022-10130-9.

[35] Mairieli Wessel, Joseph Vargovich, Marco Aurlio Gerosa, and Christoph Treude. GitHub Actions: the impact on the pull request process. *Empirical Software Engineering*, 28(6):1–35, 2023. doi:10.1007/s10664-023-10369-w.

[36] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, pages 356–366, 2014. doi:10.1145/2568225.2568315.

[37] Dariclio Moreira Soares, Manoel Limeira de Lima Jnior, Leonardo Murta, and Alexandre Plastino. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC 2015)*, pages 1541–1546, 2015. doi:10.1145/2695664.2695856.

[38] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 59(8):1–14, 2016. doi:10.1007/s11432-016-5595-8.

[39] Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart de Water. Studying pull request merges: a case study of Shopify's Active Merchant. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2018)*, pages 124–133, 2018. doi:10.1145/3183519.3183542.

[40] Gustavo Pinto, Luiz Felipe Dias, and Igor Steinmacher. Who gets a patch accepted first? Comparing the contributions of employees and volunteers. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2018)*, pages 110–113, 2018. doi:10.1145/3195836.3195858.

[41] Weiqin Zou, Jifeng Xuan, Xiaoyuan Xie, Zhenyu Chen, and Baowen Xu. How does code style

inconsistency affect pull request integration? An exploratory study on 117 GitHub projects. *Empirical Software Engineering*, 24(6):3871–3903, 2019. doi:10.1007/s10664-019-09720-x.

[42] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimki, and Davide Taibi. Does code quality affect pull request acceptance? An empirical study. *Journal of Systems and Software*, 171:1–14, 2021. doi:10.1016/j.jss.2020.110806.

[43] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. Gender differences and bias in open source: pull request acceptance of women versus men. *PeerJ Computer Science*, 2017(3):1–30, 2017. doi:10.7717/peerj-cs.111.

[44] Ayushi Rastogi. Do biases related to geographical location influence work-related decisions in GitHub? In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE-C 2016)*, pages 665–667, 2016. doi:10.1145/2889160.2891035.

[45] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and Andr van der Hoek. Relationship between geographical location and evaluation of developer contributions in GitHub. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2018)*, pages 1–8, 2018. doi:10.1145/3239235.3240504.

[46] Reza Nadri, Gema Rodrguez-Prez, and Meiyappan Nagappan. Insights into nonmerged pull requests in GitHub: is there evidence of bias based on perceptible race? *IEEE Software*, 38(2):51–57, 2021. doi:10.1109/MS.2020.3036758.

[47] Reza Nadri, Gema Rodrguez-Prez, and Meiyappan Nagappan. On the relationship between the developer's perceptible race and ethnicity and the evaluation of contributions in OSS. *IEEE Transactions on Software Engineering*, 48(8):2955–2968, 2022. doi:10.1109/TSE.2021.3073773.

[48] Leonardo B. Furtado, Bruno Cartaxo, Christoph Treude, and Gustavo Pinto. How successful are open source contributions from countries with different levels of human development? *IEEE Software*, 38 (2):58–63, 2021. doi:10.1109/MS.2020.3044020.

[49] Rahul N. Iyer, S. Alex Yun, Meiyappan Nagappan, and Jesse Hoey. Effects of personality traits on pull request acceptance. *IEEE Transactions on Software Engineering*, 47(11):2632–2643, 2021. doi:10.1109/TSE.2019.2960357.

[50] Colin G. DeYoung, Lena C. Quilty, and Jordan B. Peterson. Between facets and domains: 10 aspects of the Big Five. *Journal of Personality and Social Psychology*, 93(5):880–896, 2007. doi:10.1037/0022-3514.93.5.880.

[51] Gunnar Kudrjavets, Aditya Kumar, Nachiappan Nagappan, and Ayushi Rastogi. Mining code review data to understand waiting times between acceptance and merging: an empirical analysis. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022)*, pages 579–590, 2022. doi:10.1145/3524842.3528432.

[52] Zhixing Li, Yue Yu, Minghui Zhou, Tao Wang, Gang Yin, Long Lan, and Huaimin Wang. Redundancy, context, and preference: an empirical study of duplicate pull requests in OSS projects. *IEEE Transactions on Software Engineering*, 48(4):1309–1335, 2022. doi:10.1109/TSE.2020.3018726.

[53] Yue Yu, Zhixing Li, Gang Yin, Tao Wang, and Huaimin Wang. A dataset of duplicate pull-requests in GitHub. In *International Conference on Mining Software Repositories (MSR)*, pages 22–25, 2018. doi:10.1145/3196398.3196455.

[54] Zhixing Li, Gang Yin, Yue Yu, Tao Wang, and Huaimin Wang. Detecting duplicate pull-requests in GitHub. In *Asia-Pacific Symposium on Internetware (Internetware)*, pages 1–6, 2017. doi:10.1145/3131704.3131725.

[55] Zhi-Xing Li, Yue Yu, Tao Wang, Gang Yin, Xin-Jun Mao, and Huai-Min Wang. Detecting duplicate contributions in pull-based model combining textual and change similarities. *Journal of Computer Science and Technology*, 36(1):191–206, 2021. doi:10.1007/s11390-020-9935-1.

[56] Luyao Ren, Shurui Zhou, Christian Kstner, and Andrzej Wsowski. Identifying redundancies in fork-based development. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 230–241, 2019. doi:10.1109/SANER.2019.8668023.

[57] Qingye Wang, Bowen Xu, Xin Xia, Ting Wang, and Shanping Li. Duplicate pull request detection: when time matters. In *Asia-Pacific Symposium on Internetware (Internetware)*, pages 1–10, 2019. doi:10.1145/3361242.3361254.

[58] John W. Creswell and J. David Creswell. *Research Design: Qualitative, Quantitative, and Mixed*

Methods Approaches. SAGE Publications, Inc, 5th edition, 2017. URL https://us.sagepub.com/en-us/nam/research-design/book255675.

[59] GitHub. The state of the Octoverse, 2022. URL https://octoverse.github.com.

[60] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, 21(5):2035–2071, 2016. doi:10.1007/s10664-015-9393-5.

[61] Hudson Borges and Marco Tulio Valente. What's in a GitHub star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018. doi:10.1016/j.jss.2018.09.016.

[62] Vincent Jacques. PyGithub: typed interactions with the GitHub API v3, 2023. URL https://github.com/PyGithub/PyGithub.

[63] GitHub. Issues - GitHub Docs, 2023. URL https://docs.github.com/en/rest/reference/issues.

[64] GitHub. Pulls - GitHub Docs, 2023. URL https://docs.github.com/en/rest/reference/pulls.

[65] GitHub. REST API - GitHub Docs, 2023. URL https://docs.github.com/en/rest.

[66] Georgios Gousios. The GHTorent dataset and tool suite. In *Working Conference on Mining Software Repositories (MSR)*, pages 233–236, 2013. doi:10.1109/MSR.2013.6624034.

[67] Ilya Grigorik. GH Archive: A project to record the public GitHub timeline, archive it, and make it easily accessible for further analysis, 2023. URL https://www.gharchive.org.

[68] SayedHassan Khatoonabadi, Shahriar Lotfi, and Ayaz Isazadeh. GAP2WSS: a genetic algorithm based on the Pareto principle for web service selection, 2021.

[69] GitHub. Enums - GitHub Docs, 2022. URL https://docs.github.com/en/graphql/reference/enums.

[70] Callum Macrae. Pull Request #4781 - Homebrew/homebrew-cask, 2014. URL https://github.com/Homebrew/homebrew-cask/pull/4781.

[71] Georgios Gousios and Andy Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*, pages 368–371, 2014. doi:10.1145/2597073.2597122.

[72] Xunhui Zhang, Ayushi Rastogi, and Yue Yu. On the shoulders of giants: a new dataset for pull-based development research. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020)*, pages 543–547, 2020. doi:10.1145/3379597.3387489.

[73] Jerry L. Hintze and Ray D. Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998. doi:10.1080/00031305.1998.10480559.

[74] Indrajeet Patil. Visualizations with statistical details: the 'ggstatsplot' approach. *Journal of Open Source Software*, 6(61):1–5, 2021. doi:10.21105/joss.03167.

[75] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947. doi:10.1214/aoms/1177730491.

[76] R Core Team. R: a language and environment for statistical computing, 2023. URL https://www.R-project.org.

[77] Roger E. Kirk. Practical significance: a concept whose time has come. *Educational and Psychological Measurement*, 56(5):746–759, 1996. doi:10.1177/0013164496056005002.

[78] Norman Cliff. Dominance statistics: ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3):494–509, 1993. doi:10.1037/0033-2909.114.3.494.

[79] Mattan S. Ben-Shachar, Daniel Ldecke, and Dominique Makowski. effectsize: estimation of effect size indices and standardized parameters. *Journal of Open Source Software*, 5(56):1–7, 2020. doi:10.21105/joss.02815.

[80] Melinda R. Hess and Jeffrey D. Kromrey. Robust confidence intervals for effect sizes: a comparative study of Cohen's d and Cliff's delta under non-normality and heterogeneous variances. In *Presented at the Annual Meeting of the American Educational Research Association (AERA 2004)*, pages 1–30, 2004. URL https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.487.8299.

[81] Orta Therox. Changes to how we manage DefinitelyTyped, 2020. URL https://devblogs.microsoft.com/typescript/changes-to-how-we-manage-definitelytyped.

[82] Carsten F. Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel Carr, Jaime R. Garca Marquz, Bernd Gruber, Bruno Lafourcade, Pedro J. Leito, Tamara Mnkemller, Colin McClean, Patrick E. Osborne, Bjrn Reineking, Boris Schrder, Andrew K. Skidmore, Damaris Zurell, and Sven Lautenbach. Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. *Ecography*, 36(1):27–46, 2013. doi:10.1111/j.1600-0587.2012.07348.x.

[83] Charles Spearman. The proof and measurement of association between two things. *International Journal of Epidemiology*, 39(5):1137–1150, 2010. doi:10.1093/ije/dyq191.

[84] Frank E. Harrell. Hmisc: Harrell Miscellaneous, 2021. URL https://CRAN.R-project.org/package=Hmisc.

[85] James D. Evans. *Straightforward Statistics for the Behavioral Sciences*. Thomson Brooks/Cole Publishing Co., 1996. URL https://psycnet.apa.org/record/1995-98499-000.

[86] Marvin N. Wright and Andreas Ziegler. ranger: a fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017. doi:10.18637/jss.v077.i01.

[87] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi:10.1023/A:1010933404324.

[88] Philipp Probst, Marvin N. Wright, and AnneLaure Boulesteix. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery*, 9(3):e1301, 2019. doi:10.1002/widm.1301.

[89] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998. doi:10.1023/A:1008306431147.

[90] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. doi:10.1109/TKDE.2008.239.

[91] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997. doi:10.1016/S0031-3203(96)00142-2.

[92] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. doi:10.1016/j.patrec.2005.10.010.

[93] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL https://jmlr.org/papers/v17/15-066.html.

[94] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019. URL https://jmlr.org/papers/v20/18-760.html.

[95] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. iml: an R package for interpretable machine learning. *Journal of Open Source Software*, 3(27):786, 2018. doi:10.21105/joss.00786.

[96] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4): 1059–1086, 2020. doi:10.1111/rssb.12377.

[97] Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2nd edition, 2022. URL https://christophm.github.io/interpretable-ml-book.

[98] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999. doi:10.1109/32.799955.

[99] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. doi:10.1177/001316446002000104.

[100] Fabian Pedregosa, Gal Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and douard Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL https://jmlr.org/papers/v12/pedregosa11a.html.

[101] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977. doi:10.2307/2529310.

[102] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Studying software engineers: data collection techniques for software field studies. *Empirical Software Engineering*, 10(3):311–341, 2005. doi:10.1007/s10664-005-1290-x.

[103] Kubernetes. fejta-bot GitHub user, 2021. URL https://github.com/fejta-bot.

[104] Igor Steinmacher, Igor Wiese, Ana Paula Chaves, and Marco Aurlio Gerosa. Why do newcomers abandon open source software projects? In *Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2013)*, pages 25–32, 2013. doi:10.1109/CHASE.2013.6614728.

[105] Igor Steinmacher, Ana Paula Chaves, Tayana Uchoa Conte, and Marco Aurlio Gerosa. Preliminary empirical identification of barriers faced by newcomers to open source software projects. In *Proceedings of the Brazilian Symposium on Software Engineering (SBES 2014)*, pages 51–60, 2014. doi:10.1109/SBES.2014.9.

[106] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurlio Gerosa. Almost there: a study on quasi-contributors in open source software projects. In *International Conference on Software Engineering (ICSE)*, pages 256–266, 2018. doi:10.1145/3180155.3180208.

[107] Qingye Wang, Xin Xia, David Lo, and Shanping Li. Why is my code change abandoned? *Information and Software Technology*, 110:108–120, 2019. doi:10.1016/j.infsof.2019.02.007.

[108] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. Chapman & Hall/CRC, 2015. doi:10.1201/b19467.

[109] Igor Steinmacher, Tayana Conte, Marco Aurlio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW 2015)*, pages 1379–1392, 2015. doi:10.1145/2675133.2675215.

[110] Minghui Zhou and Audris Mockus. What make long term contributors: willingness and opportunity in OSS community. In *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, pages 518–528, 2012. doi:10.1109/ICSE.2012.6227164.

[111] GitHub. Events - GitHub Docs, 2023. URL https://docs.github.com/en/rest/activity/events.

[112] Google. BigQuery: cloud data warehouse, 2023. URL https://cloud.google.com/bigquery.

[113] GitHub. Timeline - GitHub Docs, 2023. URL https://docs.github.com/en/rest/issues/timeline.

[114] Brady T. West, Kathleen B. Welch, and Andrzej T. Gaecki. *Linear Mixed Models: A Practical Guide Using Statistical Software*. Chapman & Hall/CRC, 2nd edition, 2014. doi:10.1201/b17198.

[115] Andrzej Gaecki and Tomasz Burzykowski. *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*. Springer Texts in Statistics. Springer, 2013. doi:10.1007/978-1-4614-3900-4.

[116] Alexandra Kuznetsova, Per B. Brockhoff, and Rune H. B. Christensen. lmerTest package: tests in linear mixed effects models. *Journal of Statistical Software*, 82(13):1–26, 2017. doi:10.18637/jss.v082.i13.

[117] Shinichi Nakagawa, Paul C. D. Johnson, and Holger Schielzeth. The coefficient of determination R^2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. *Journal of The Royal Society Interface*, 14(134):1–11, 2017. doi:10.1098/rsif.2017.0213.

[118] Daniel Ldecke, Mattan S. Ben-Shachar, Indrajeet Patil, Philip Waggoner, and Dominique Makowski. performance: an R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software*, 6(60):1–8, 2021. doi:10.21105/joss.03139.

[119] Alex Hrong-Tai Fai and Paul L. Cornelius. Approximate F-tests of multiple degree of freedom hypotheses in generalized least squares analyses of unbalanced split-plot experiments. *Journal of Statistical Computation and Simulation*, 54(4):363–378, 1996. doi:10.1080/00949659608811740.

[120] David Chavalarias, Joshua David Wallach, Alvin Ho Ting Li, and John P. A. Ioannidis. Evolution of reporting *P* values in the biomedical literature, 1990-2015. *JAMA*, 315(11):1141–1148, 2016. doi:10.1001/jama.2016.1952.

[121] James Howard Goodnight. Tests of hypotheses in fixed effects linear models. *Communications in Statistics - Theory and Methods*, 9(2):167–180, 1980. doi:10.1080/03610928008827869.

[122] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. The SPACE of developer productivity: there's more to it than you think. *Queue*, 19(1): 20–48, 2021. doi:10.1145/3454122.3454124.

[123] Alexander Ziborov. Commit #439b8735fa93709ec32602bb32944bf9214ce785 - DevExpress/DevExtreme, 2020. URL https://github.com/DevExpress/DevExtreme/commit/439b8735fa93709ec32602bb32944bf9214ce785.

[124] Manoel Limeira de Lima Jnior, Dariclio Soares, Alexandre Plastino, and Leonardo Murta. Predicting the lifetime of pull requests in open-source projects. *Journal of Software: Evolution and Process*, 33 (6):1–23, 2021. doi:10.1002/smr.2337.

[125] Chandra Maddila, Chetan Bansal, and Nachiappan Nagappan. Predicting pull request completion time: a case study on large scale cloud services. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*, pages 874–882, 2019. doi:10.1145/3338906.3340457.

[126] Chandra Maddila, Sai Surya Upadrasta, Chetan Bansal, Nachiappan Nagappan, Georgios Gousios, and Arie van Deursen. Nudge: accelerating overdue pull requests towards completion. *ACM Transactions on Software Engineering and Methodology*, 32(2):1–30, 2023. doi:10.1145/3544791.

[127] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NIPS 2017)*, volume 30, pages 1–10, 2017.

[128] Deepika Badampudi, Michael Unterkalmsteiner, and Ricardo Britto. Modern code reviewssurvey of literature and practice. *ACM Transactions on Software Engineering and Methodology*, 32(4):1–61, 2023. doi:10.1145/3585004.

[129] Nicole Davila and Ingrid Nunes. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software*, 177:1–30, 2021. doi:10.1016/j.jss.2021.110951.

[130] Thomas Bock, Nils Alznauer, Mitchell Joblin, and Sven Apel. Automatic core-developer identification on GitHub: a validation study. *ACM Transactions on Software Engineering and Methodology*, 32(6): 1–29, 2023. doi:10.1145/3593803.

[131] Ahmad Abdellatif, Mairieli Wessel, Igor Steinmacher, Marco A. Gerosa, and Emad Shihab. BotHunter: an approach to detect software bots in GitHub. In *Proceedings of the IEEE/ACM 19th International Conference on Mining Software Repositories (MSR 2022)*, pages 6–17, 2022. doi:10.1145/3524842.3527959.

[132] Zhendong Wang, Yi Wang, and David Redmiles. From specialized mechanics to project butlers: the usage of bots in open source software development. *IEEE Software*, 39(5):38–43, 2022. doi:10.1109/MS.2022.3180297.

[133] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software*, 175:1–14, 2021. doi:10.1016/j.jss.2021.110911.

[134] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. Review participation in modern code review: an empirical study of the Android, Qt, and OpenStack projects. *Empirical Software Engineering*, 22(2):768–817, 2017. doi:10.1007/s10664-016-9452-6.

[135] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Ahmed E. Hassan. The impact of correlated metrics on the interpretation of defect models. *IEEE Transactions on Software Engineering*, 47(2): 320–331, 2021. doi:10.1109/TSE.2019.2891758.

[136] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31, pages 1–11, 2018.

[137] Jafar Tanha, Yousef Abdi, Negin Samadi, Nazila Razzaghi, and Mohammad Asadpour. Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data*, 7 (1):1–47, 2020. doi:10.1186/s40537-020-00349-y.

[138] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. Machine/deep learning for software engineering: a systematic literature review. *IEEE Transactions on Software Engineering*, 49(3):1188–1231, 2023. doi:10.1109/TSE.2022.3173346.

[139] Yanming Yang, Xin Xia, David Lo, Tingting Bi, John Grundy, and Xiaohu Yang. Predictive models in software engineering: challenges and opportunities. *ACM Transactions on Software Engineering and Methodology*, 31(3):1–72, 2022. doi:10.1145/3503509.

[140] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004. URL https://www.jmlr.org/papers/v5/rifkin04a.html.

[141] Davide DellAnna, Fatma Baak Aydemir, and Fabiano Dalpiaz. Evaluating classifiers in SE research: the ECSER pipeline and two replication studies. *Empirical Software Engineering*, 28(1):1–40, 2023. doi:10.1007/s10664-022-10243-1.

[142] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 45(7):683–711, 2019. doi:10.1109/TSE.2018.2794977.

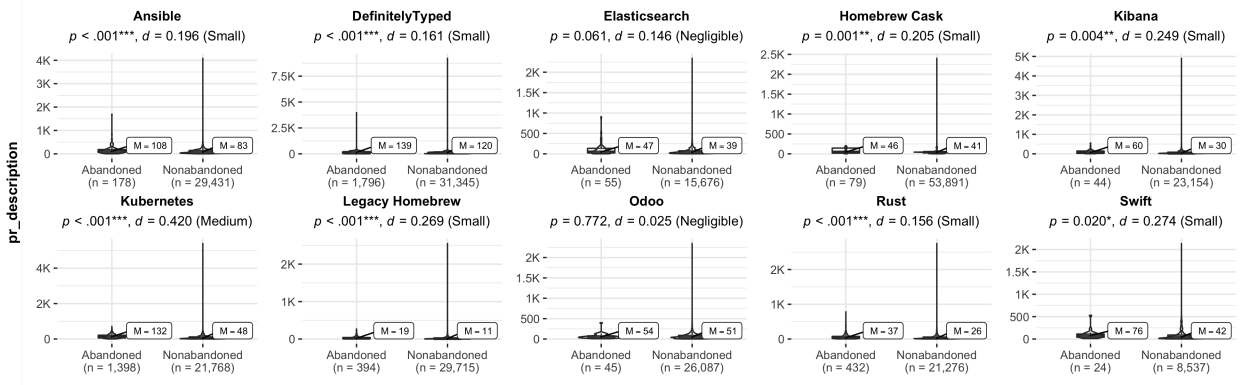
[143] Gopi Krishnan Rajbahadur, Shaowei Wang, Gustavo A. Oliva, Yasutaka Kamei, and Ahmed E. Hassan. The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering*, 48(7):2245–2261, 2022. doi:10.1109/TSE.2021.3056941.

[144] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. Investigating code review quality: do people and participation matter? In *Proceedings of the IEEE 31st International Conference on Software Maintenance and Evolution (ICSME 2015)*, pages 111–120, 2015. doi:10.1109/ICSM.2015.7332457.

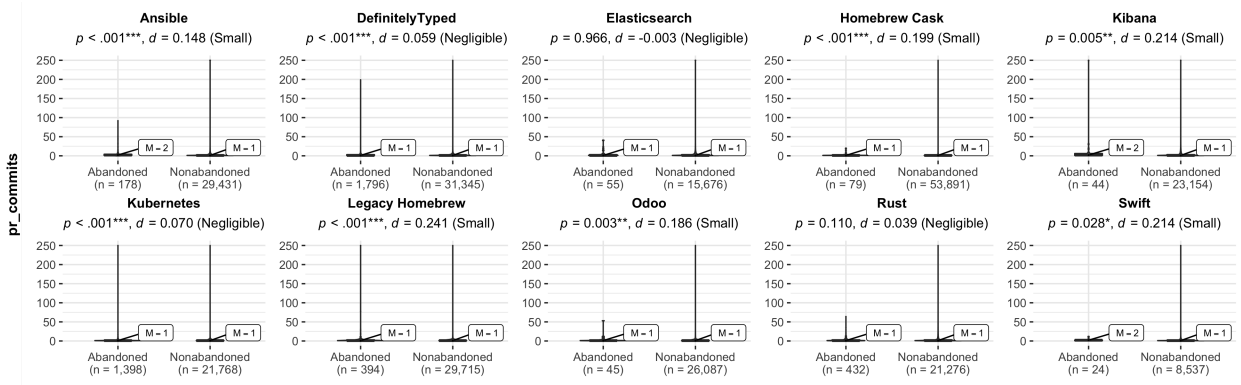
[145] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, pages 1135–1144, 2016. doi:10.1145/2939672.2939778.

Appendices

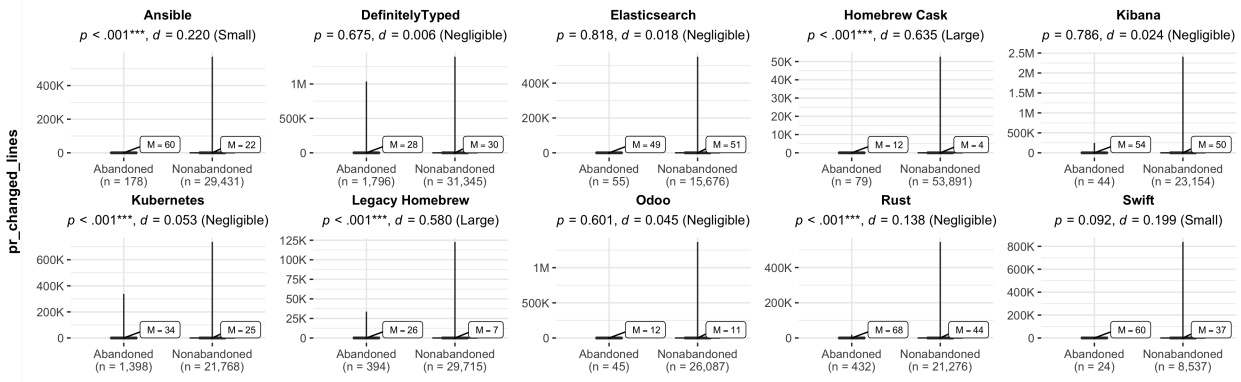
Comparison of Abandoned and Nonabandoned PRs



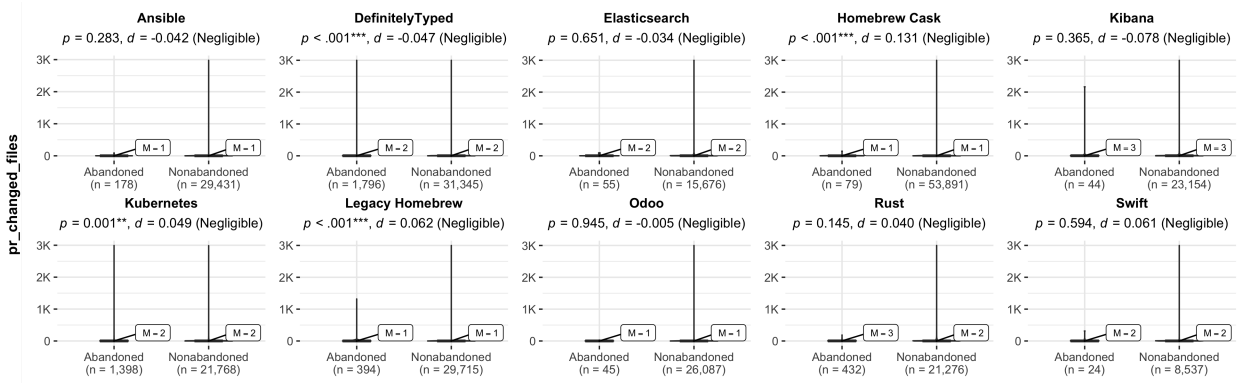
Comparison of abandoned and nonabandoned PRs wrt pr_description across the studied projects.



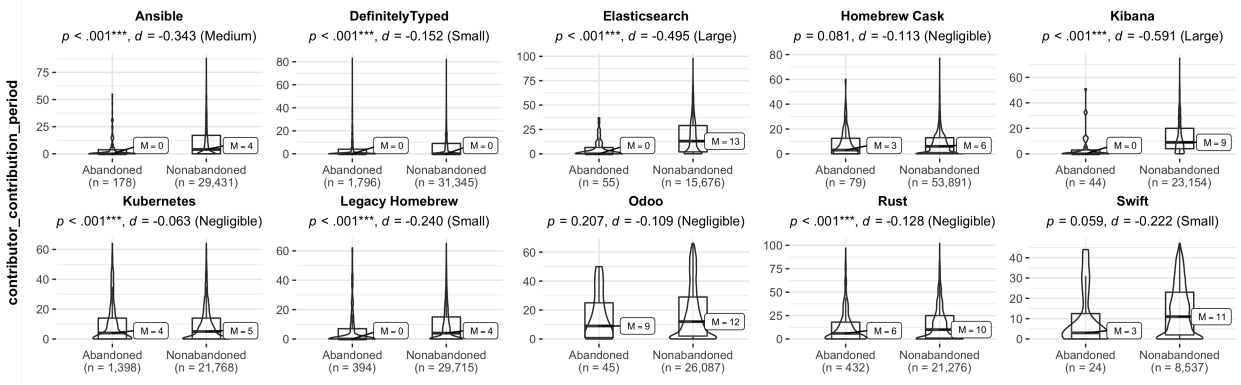
Comparison of abandoned and nonabandoned PRs wrt pr_commits across the studied projects.



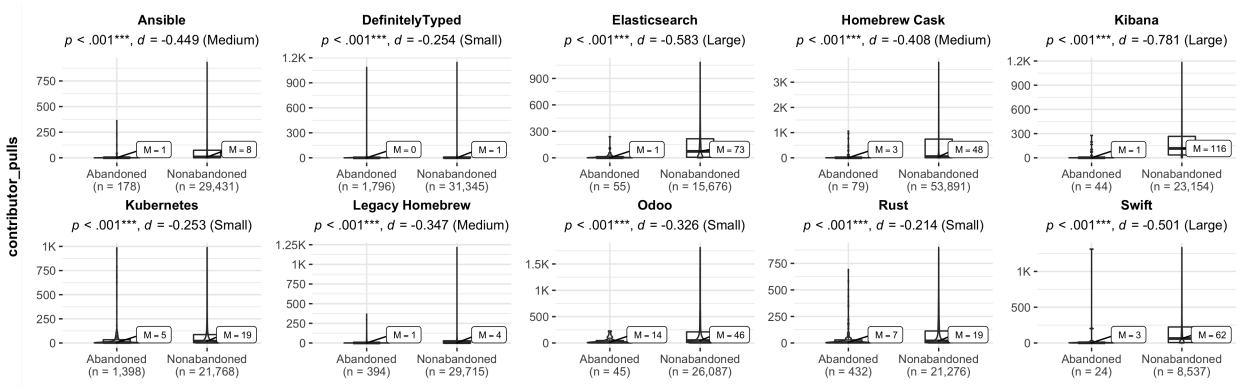
Comparison of abandoned and nonabandoned PRs wrt pr_changed_lines across the studied projects.



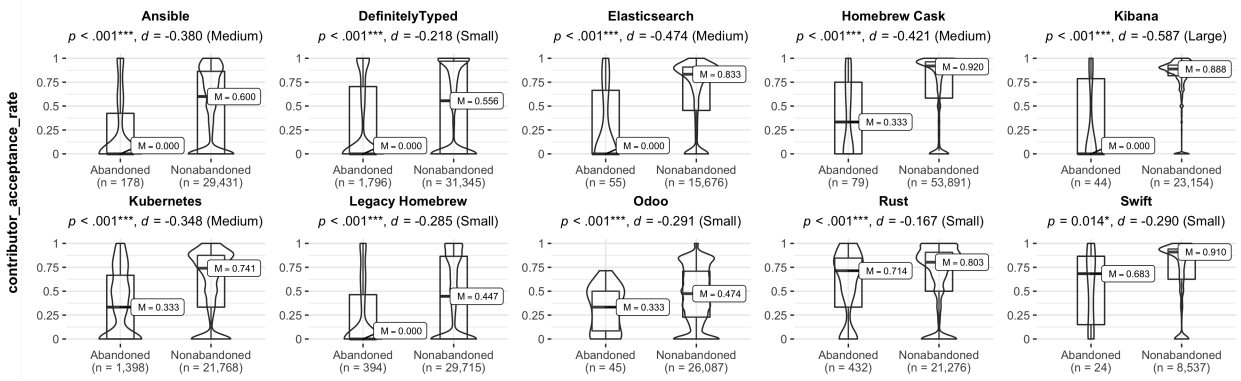
Comparison of abandoned and nonabandoned PRs wrt pr_changed_files across the studied projects.



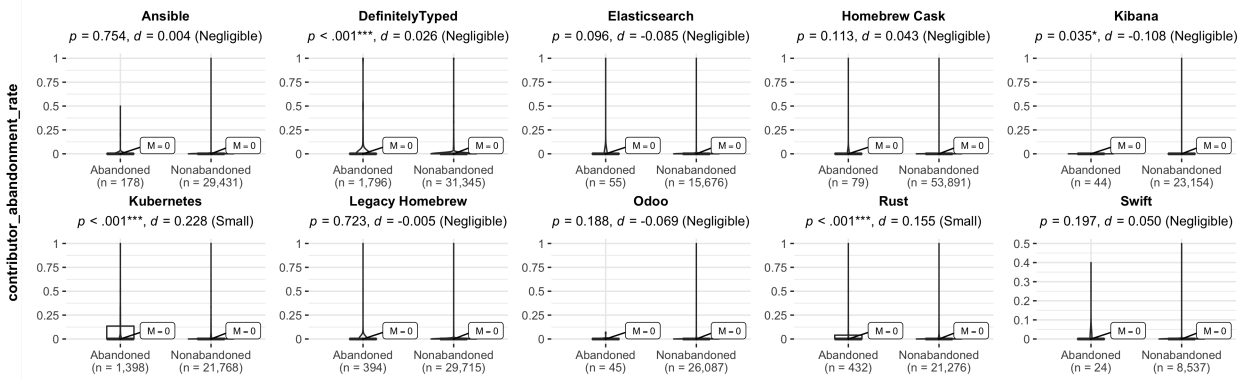
Comparison of abandoned and nonabandoned PRs wrt contributor_contribution_period across the studied projects.



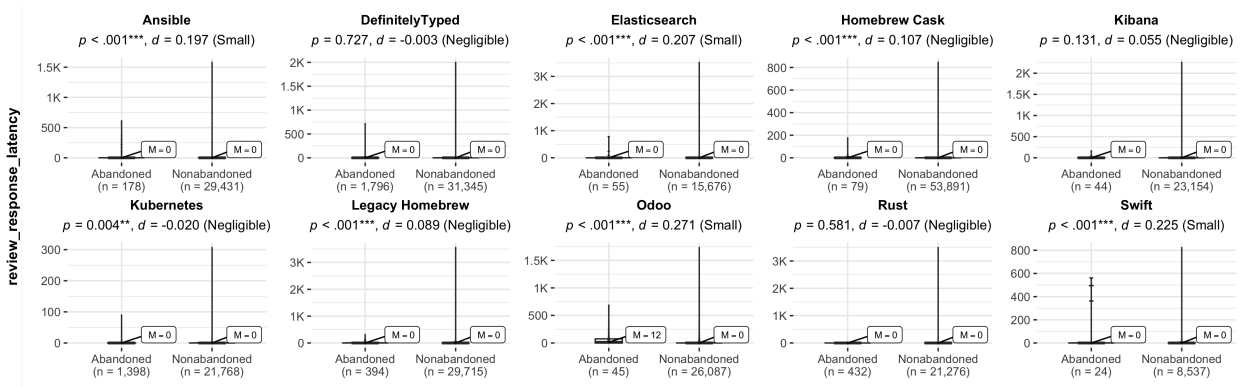
Comparison of abandoned and nonabandoned PRs wrt contributor_pulls across the studied projects.



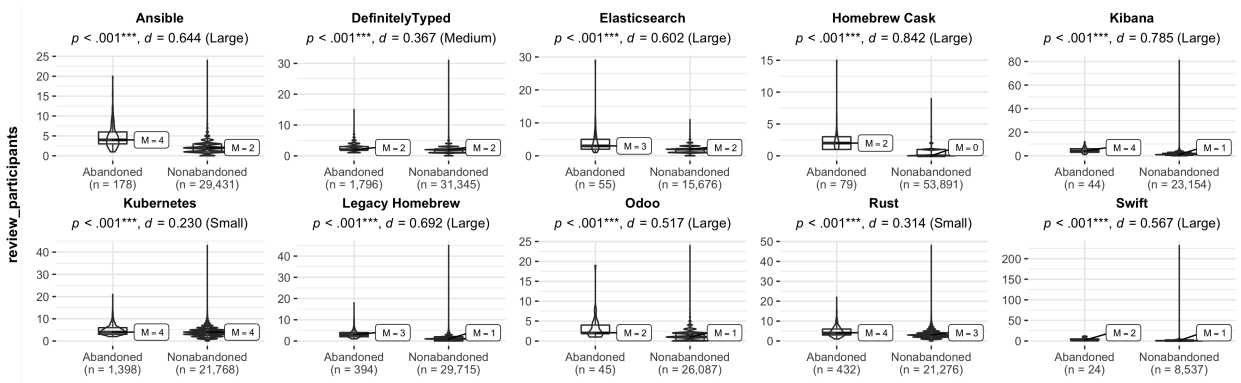
Comparison of abandoned and nonabandoned PRs wrt contributor_acceptance_rate across the studied projects.



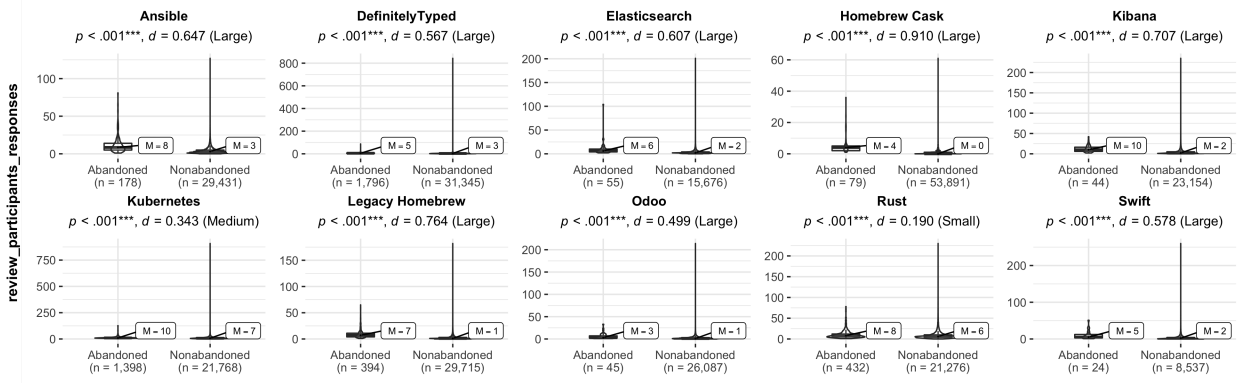
Comparison of abandoned and nonabandoned PRs wrt contributor_abandonment_rate across the studied projects.



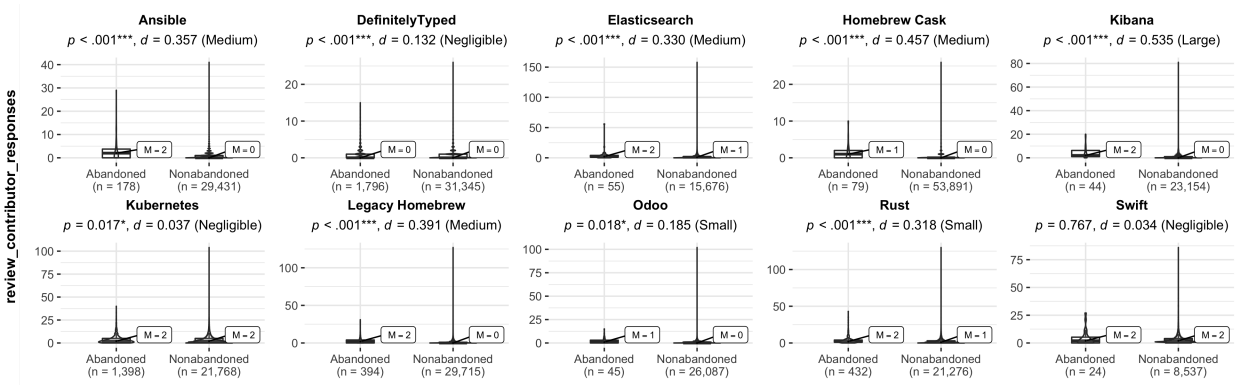
Comparison of abandoned and nonabandoned PRs wrt review_response_latency across the studied projects.



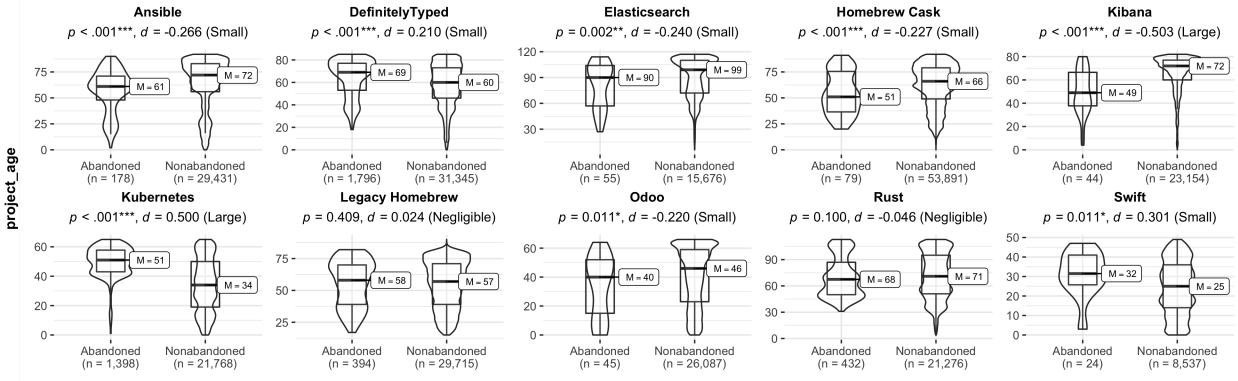
Comparison of abandoned and nonabandoned PRs wrt review_participants across the studied projects.



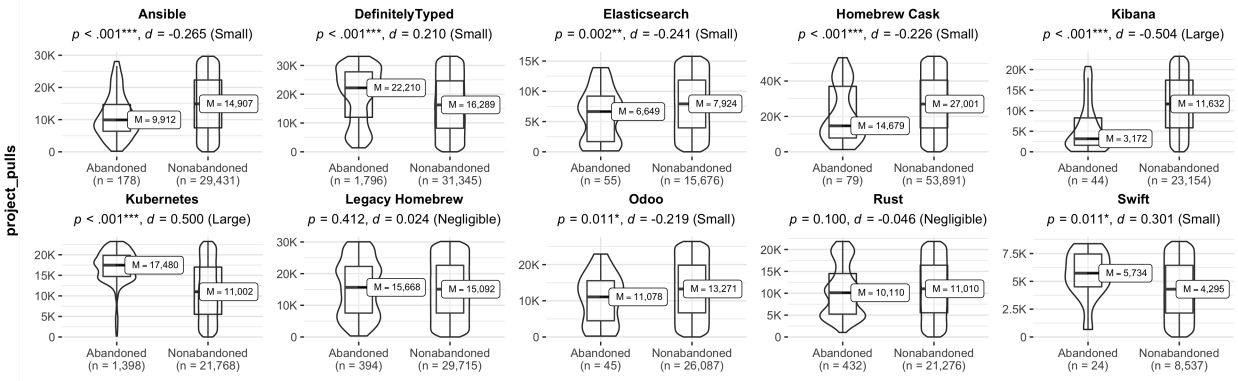
Comparison of abandoned and nonabandoned PRs wrt review_participants_responses across the studied projects.



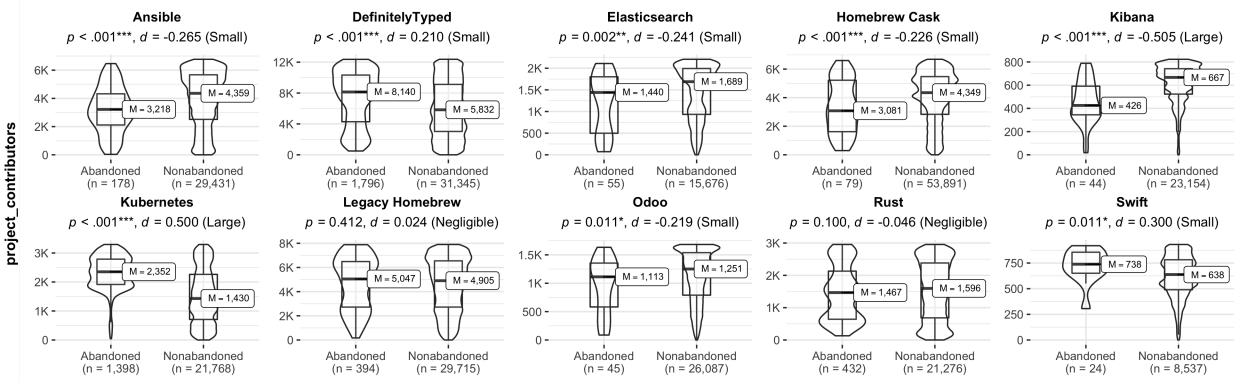
Comparison of abandoned and nonabandoned PRs wrt review_contributor_responses across the studied projects.



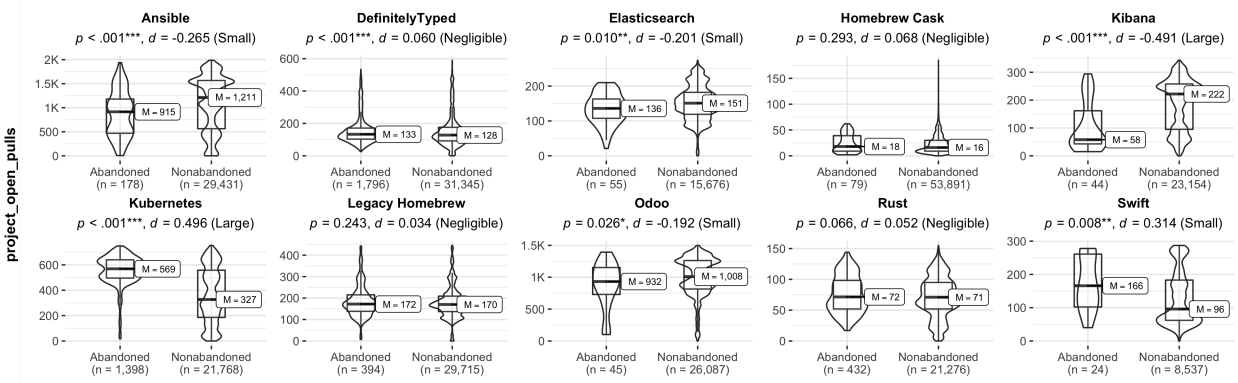
Comparison of abandoned and nonabandoned PRs wrt project_age across the studied projects.



Comparison of abandoned and nonabandoned PRs wrt project_pulls across the studied projects.

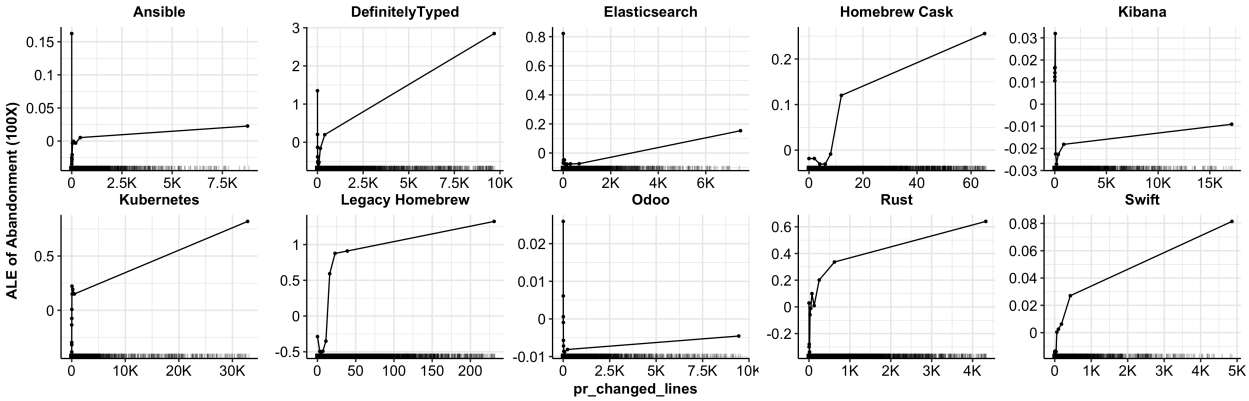


Comparison of abandoned and nonabandoned PRs wrt project_contributors across the studied projects.

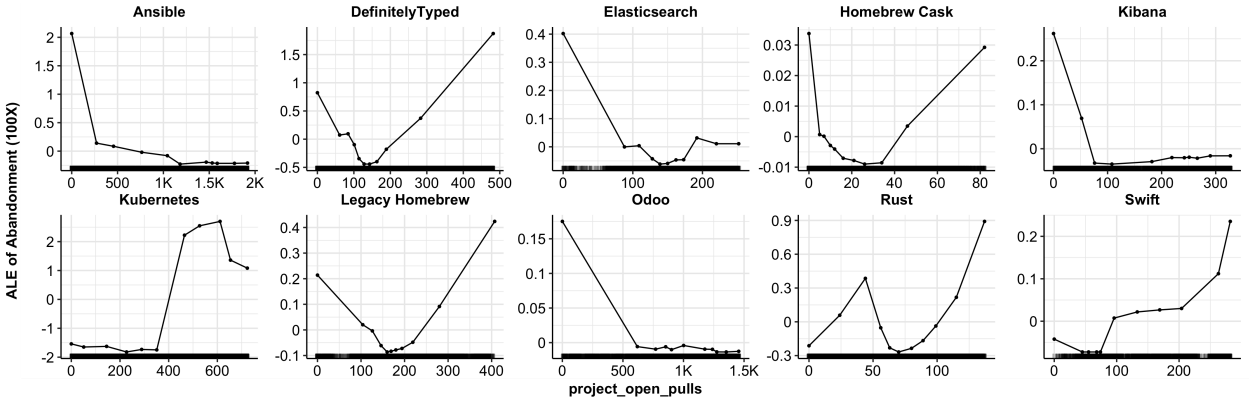


Comparison of abandoned and nonabandoned PRs wrt project_open_pulls across the studied projects.

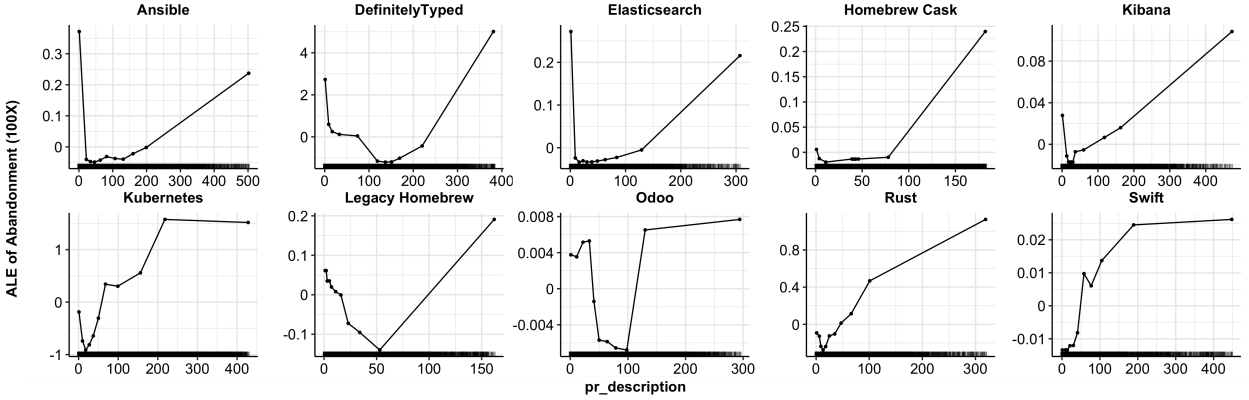
ALE Plots for Different Features



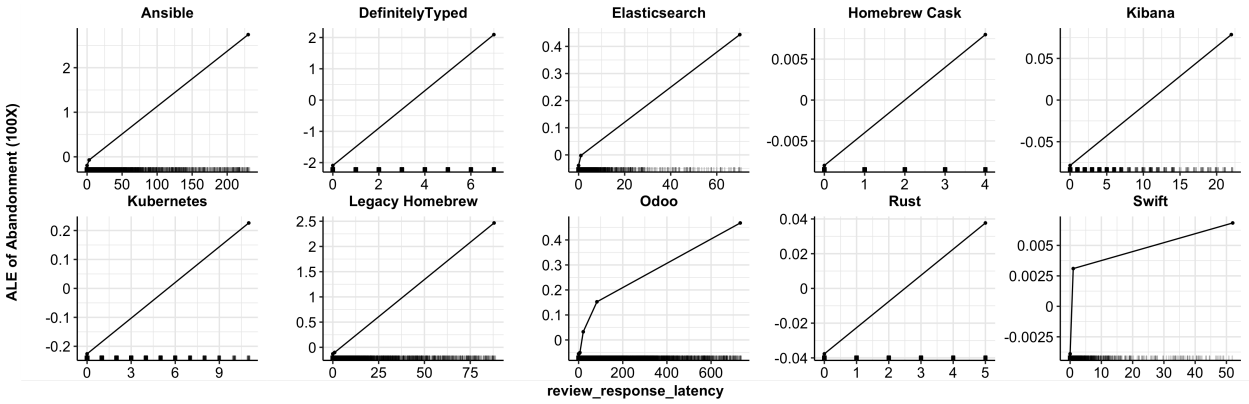
ALE plots showing how pr_changed_lines varies the abandonment probability of PRs across the studied projects.



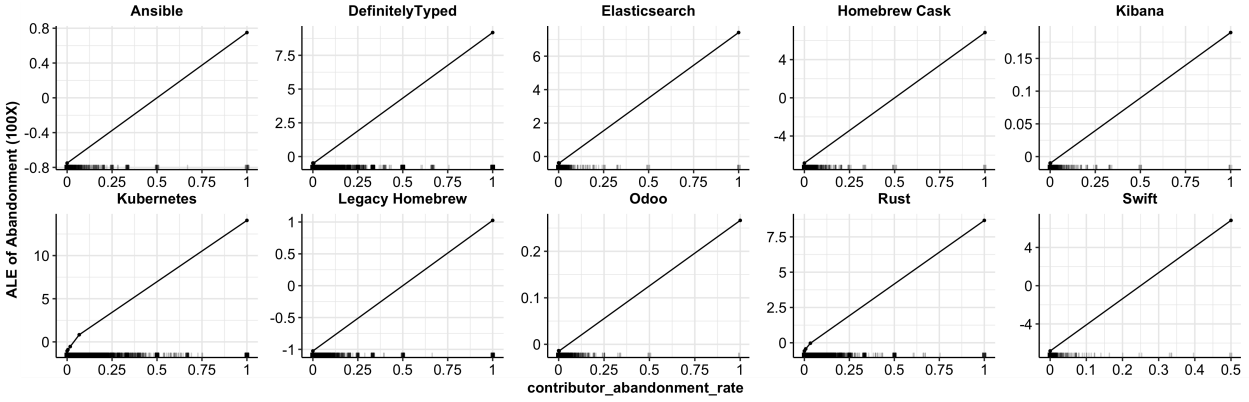
ALE plots showing how project_open_pulls varies the abandonment probability of PRs across the studied projects.



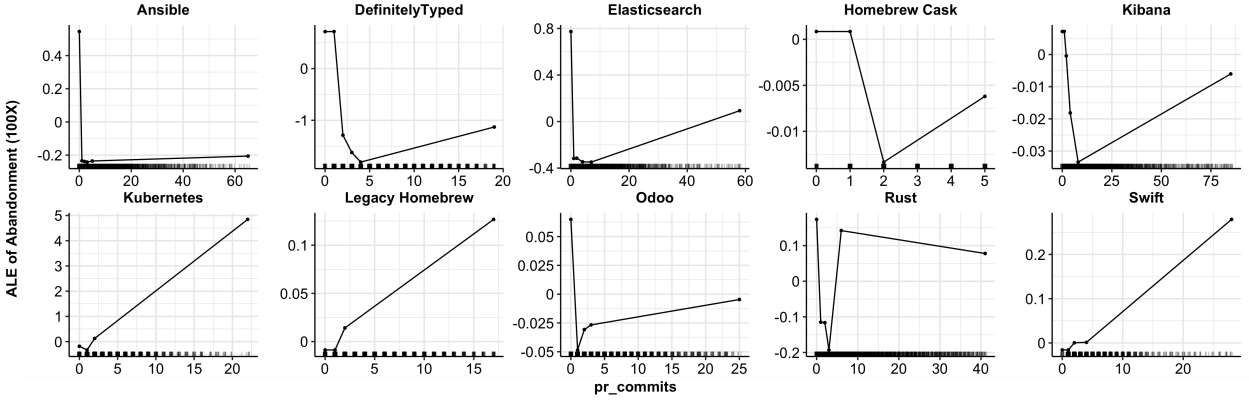
ALE plots showing how pr_description varies the abandonment probability of PRs across the studied projects.



ALE plots showing how review_response_latency varies the abandonment probability of PRs across the studied projects.



ALE plots showing how contributor_abandonment_rate varies the abandonment probability of PRs across the studied projects.



ALE plots showing how pr_commits varies the abandonment probability of PRs across the studied projects.

Models for Estimating the Impact of Stale Bot

Models for estimating the impact of adopting Stale bot on the number of merged PRs (merged_pulls) and closed PRs (closed_pulls) in the studied projects.

	log(merged_pulls)		**log(closed_pulls)**	
	Coefficient	**Sum Sq.**	**Coefficient**	**Sum Sq.**
time	0.041***	4.852***	0.038***	4.159***
adoption	-0.129*	0.496*	0.187*	1.040*
time_since_adoption	-0.021*	0.653*	-0.044**	2.721**
log(age_at_adoption)	-0.627**	1.397**	-0.678**	2.483**
log(pulls_at_adoption)	0.883***	1.792***	0.279	0.271
log(contributors_at_adoption)	0.152	0.334	0.379**	3.158**
log(maintainers_at_adoption)	-0.210	0.124	0.226	0.216
intercept	-0.688		0.299	
Marginal R^2	0.57		0.44	
Conditional R^2	0.82		0.70	

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Models for estimating the impact of adopting Stale bot on the first response latency of merged PRs (first_latency_m) and closed PRs (first_latency_c) in the studied projects.

	log(first_latency_m)		**log(first_latency_c)**	
	Coefficient	**Sum Sq.**	**Coefficient**	**Sum Sq.**
time	0.029*	2.398*	0.085***	20.577***
adoption	0.064	0.123	0.071	0.151
time_since_adoption	-0.045*	2.920*	-0.051	3.708
log(age_at_adoption)	0.983	0.786	1.162	2.381
log(pulls_at_adoption)	0.662	0.230	1.342	2.050
log(contributors_at_adoption)	-0.228	0.173	-0.748	4.027
log(maintainers_at_adoption)	-0.950	0.577	-1.257	2.186
intercept	-2.599		-4.939	
Marginal R^2	0.08		0.14	
Conditional R^2	0.87		0.77	

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Models for estimating the impact of adopting Stale bot on the mean response latency of merged PRs (mean_latency_m) and closed PRs (mean_latency_c) in the studied projects.

	log(mean_latency_m)		**log(mean_latency_c)**	
	Coefficient	**Sum Sq.**	**Coefficient**	**Sum Sq.**
time	0.011	0.318	0.031	2.728
adoption	-0.023	0.015	0.128	0.486
time_since_adoption	-0.006	0.052	-0.025	0.901
log(age_at_adoption)	0.677*	1.240*	0.770	2.698
log(pulls_at_adoption)	0.133	0.031	0.467	0.640
log(contributors_at_adoption)	0.093	0.095	-0.131	0.319
log(maintainers_at_adoption)	-0.274	0.160	-0.504	0.907
intercept	-0.158		-0.142	
Marginal R^2	0.22		0.15	
Conditional R^2	0.72		0.58	

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Models for estimating the impact of adopting Stale bot on the resolution time of merged PRs (resolution_time_m) and closed PRs (resolution_time_c) in the studied projects.

	log(resolution_time_m)		**log(resolution_time_c)**	
	Coefficient	**Sum Sq.**	**Coefficient**	**Sum Sq.**
time	-0.001	0.004	0.029*	2.366*
adoption	0.052	0.080	0.267	2.124
time_since_adoption	-0.001	0.001	-0.043*	2.602*
log(age_at_adoption)	0.509	0.460	0.547	1.007
log(pulls_at_adoption)	-0.345	0.136	-0.029	0.002
log(contributors_at_adoption)	0.228	0.376	0.156	0.336
log(maintainers_at_adoption)	0.333	0.154	0.102	0.027
intercept	3.482		3.021	
Marginal R^2	0.25		0.18	
Conditional R^2	0.80		0.66	

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Models for estimating the impact of adopting Stale bot on the number of comments in merged PRs (comments_m) and closed PRs (comments_c) in the studied projects.

	log(comments_m)		**log(comments_c)**	
	Coefficient	**Sum Sq.**	**Coefficient**	**Sum Sq.**
time	-0.009	0.216	-0.008	0.195
adoption	0.023	0.016	-0.001	0.000
time_since_adoption	0.001	0.002	-0.003	0.015
log(age_at_adoption)	-0.165	0.020	-0.075	0.009
log(pulls_at_adoption)	-0.603	0.171	-0.438	0.209
log(contributors_at_adoption)	0.221	0.145	0.300*	0.617*
log(maintainers_at_adoption)	0.826*	0.390*	0.375	0.186
intercept	3.252		2.589	
Marginal R^2	0.26		0.19	
Conditional R^2	0.90		0.79	

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Models for estimating the impact of adopting Stale bot on the number of commits in merged PRs (commits_m) and closed PRs (commits_c) in the studied projects.

	log(commits_m)		**log(commits_c)**	
	Coefficient	**Sum Sq.**	**Coefficient**	**Sum Sq.**
time	0.010	0.271	0.030*	2.501*
adoption	-0.021	0.013	-0.215	1.369
time_since_adoption	-0.015*	0.335*	-0.026	0.942
log(age_at_adoption)	0.189	0.094	0.545**	7.487**
log(pulls_at_adoption)	-0.252	0.108	-0.120	0.234
log(contributors_at_adoption)	-0.019	0.004	0.085	0.741
log(maintainers_at_adoption)	0.064	0.009	-0.188	0.700
intercept	2.580		0.783	
Marginal R^2	0.09		0.16	
Conditional R^2	0.67		0.27	

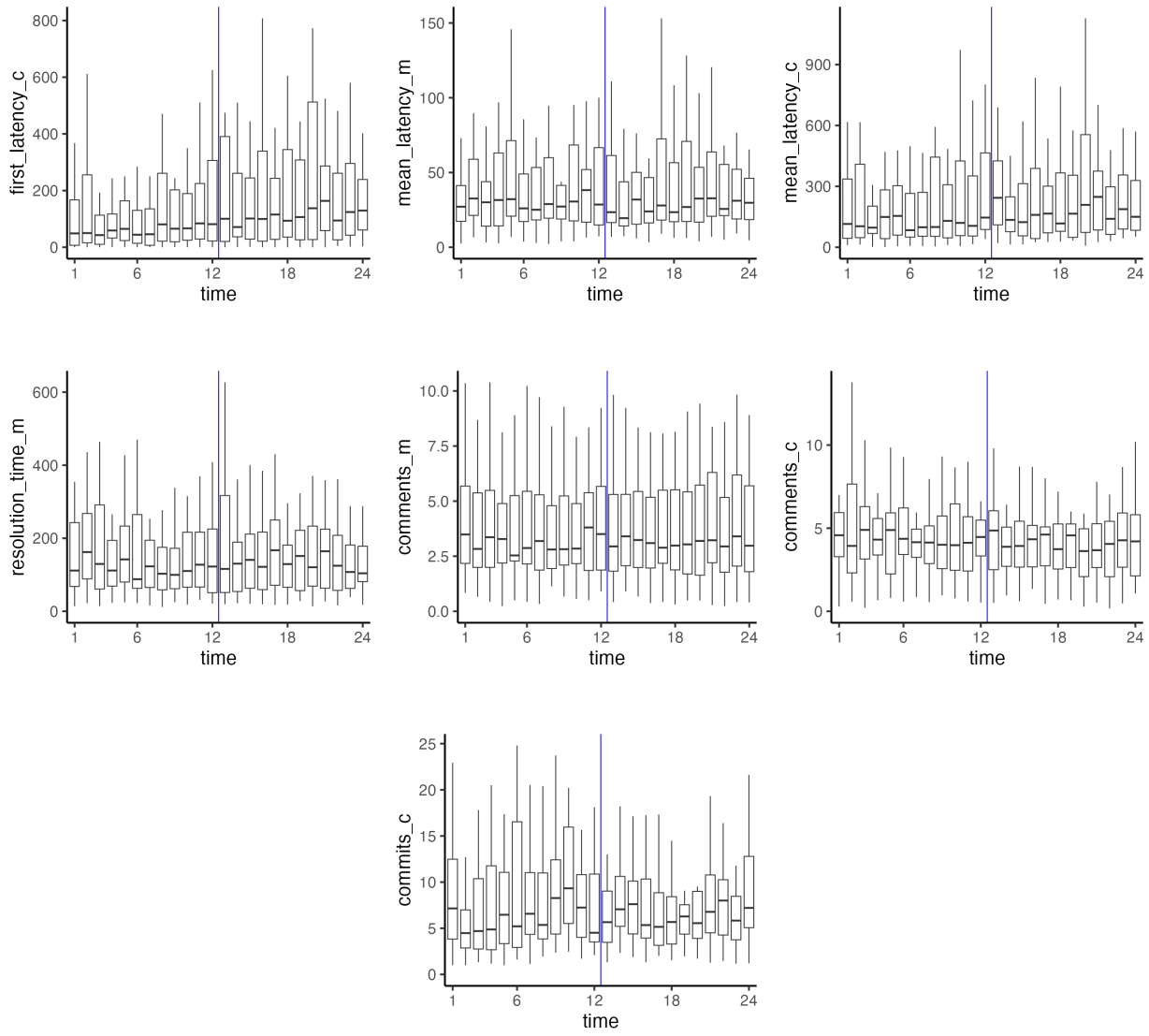
*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Models for estimating the impact of adopting Stale bot on the number of active contributors (contributors) in the studied projects.

	log(contributors)	
	Coefficient	**Sum Sq.**
time	0.026***	1.867***
adoption	0.003	0.000
time_since_adoption	-0.022***	0.708***
log(age_at_adoption)	-0.535**	0.493**
log(pulls_at_adoption)	0.250	0.070
log(contributors_at_adoption)	0.697***	3.411***
log(maintainers_at_adoption)	0.114	0.018
intercept	-1.033	
Marginal R^2	0.84	
Conditional R^2	0.96	

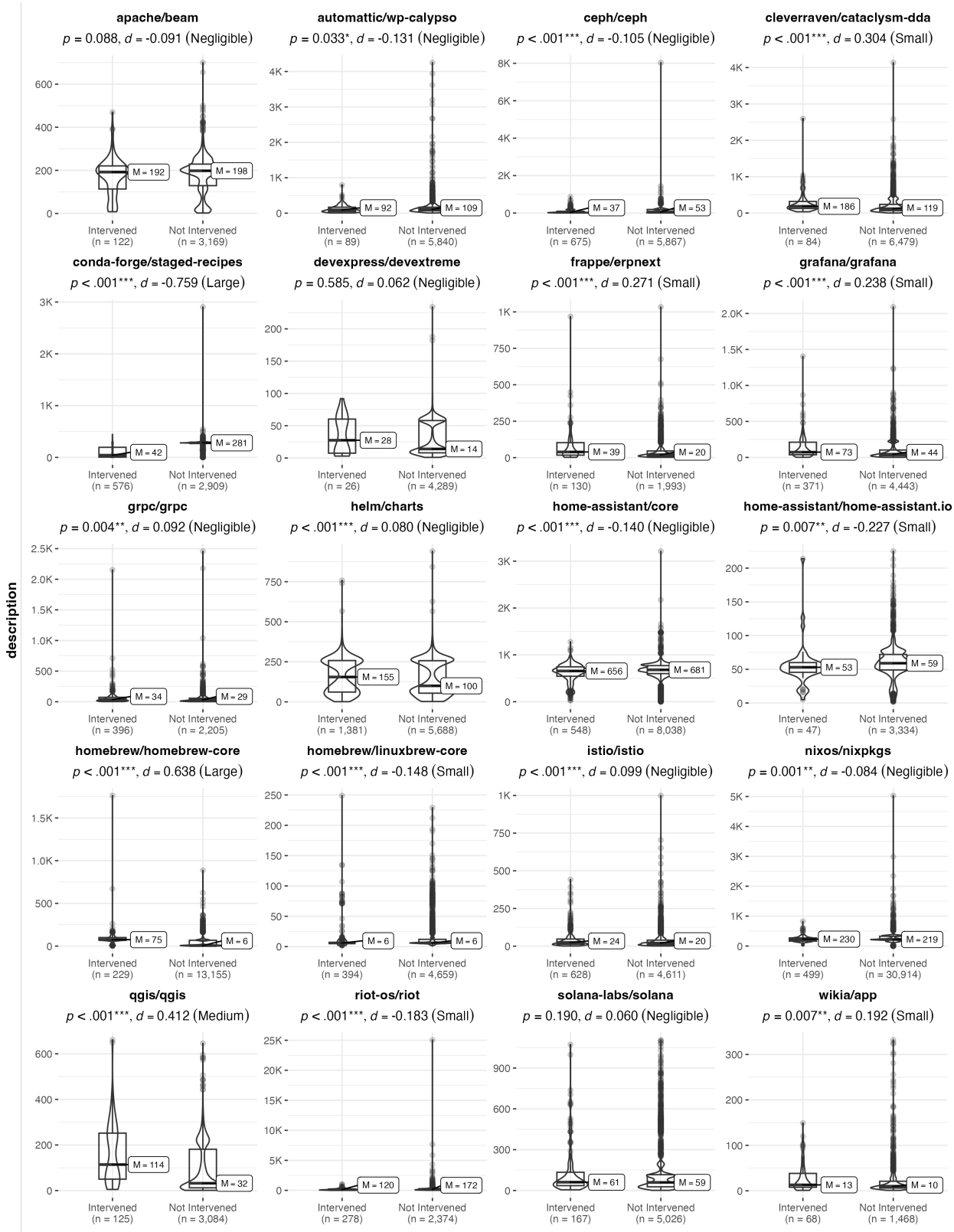
*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

Variation of the Performance Indicators

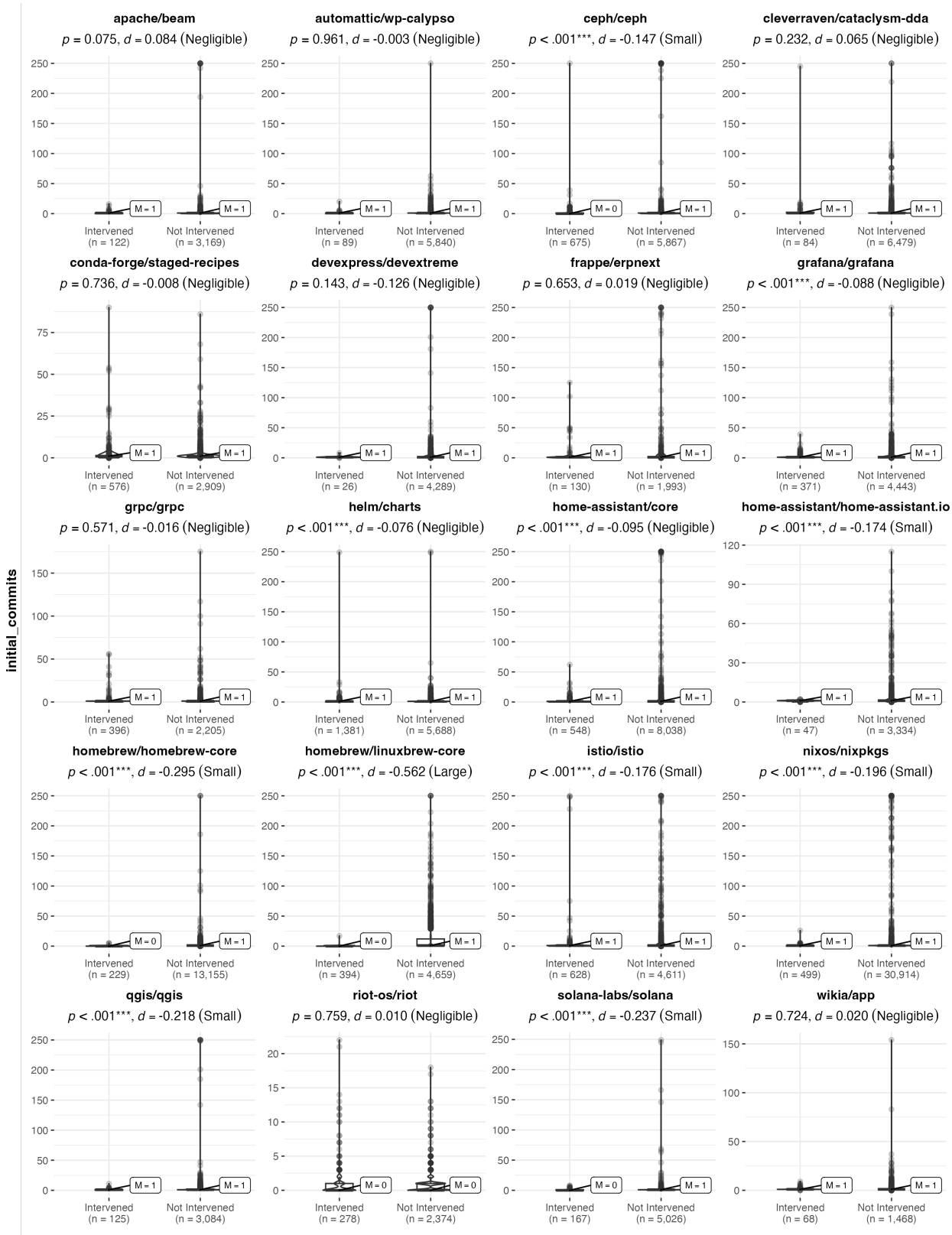


Variation in (a) the first response latency of closed PRs, (b) the mean response latency of merged PRs, (c) the mean response latency of closed PRs, (d) the resolution time of merged PRs, (e) the number of comments in merged PRs, (f) the number of comments in closed PRs, and (g) the number of commits in closed PRs each month during our observation period. The blue lines show the adoption time.

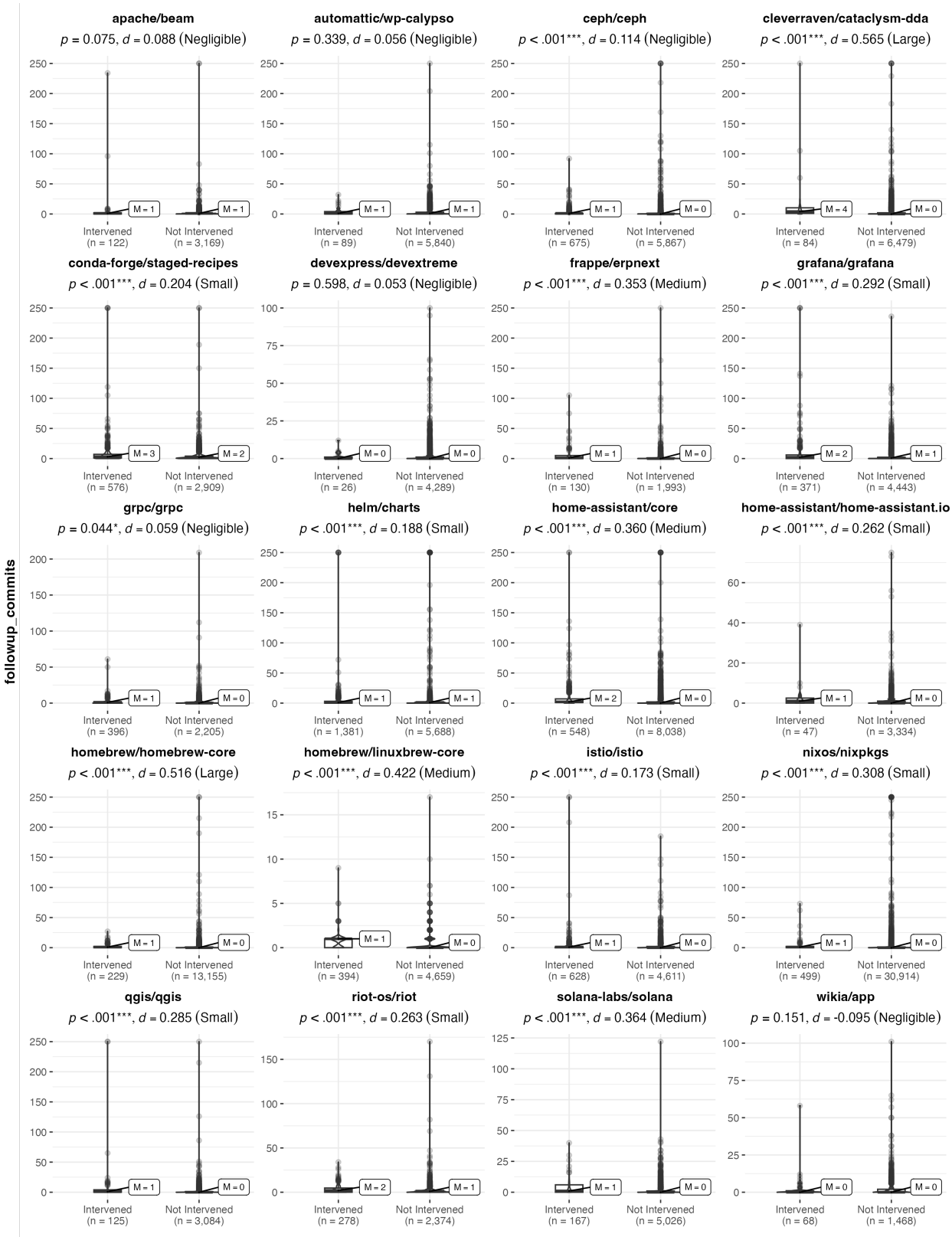
Characteristics of PRs Intervened by Stale Bot



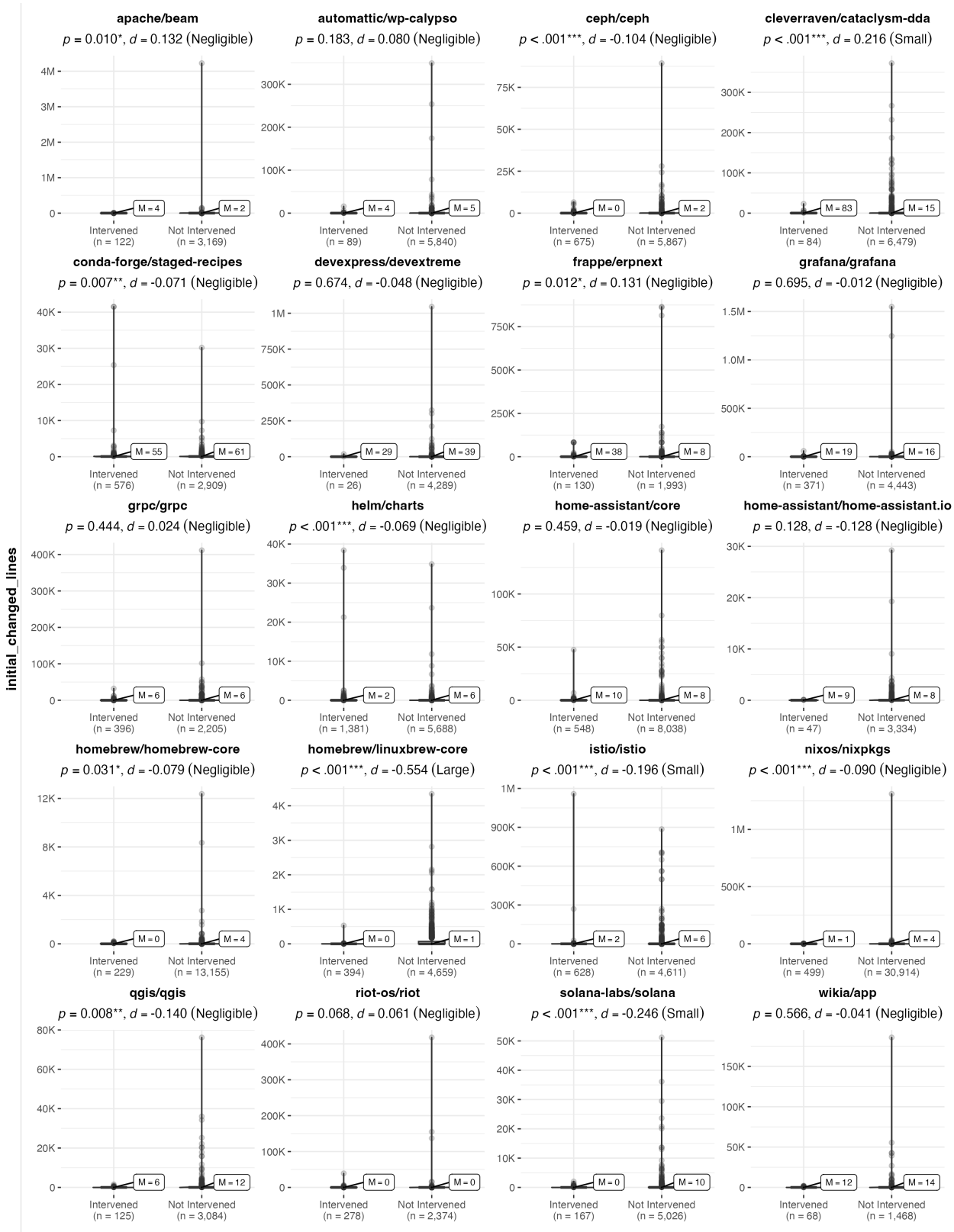
Comparison of intervened and not intervened PRs regarding their description length across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



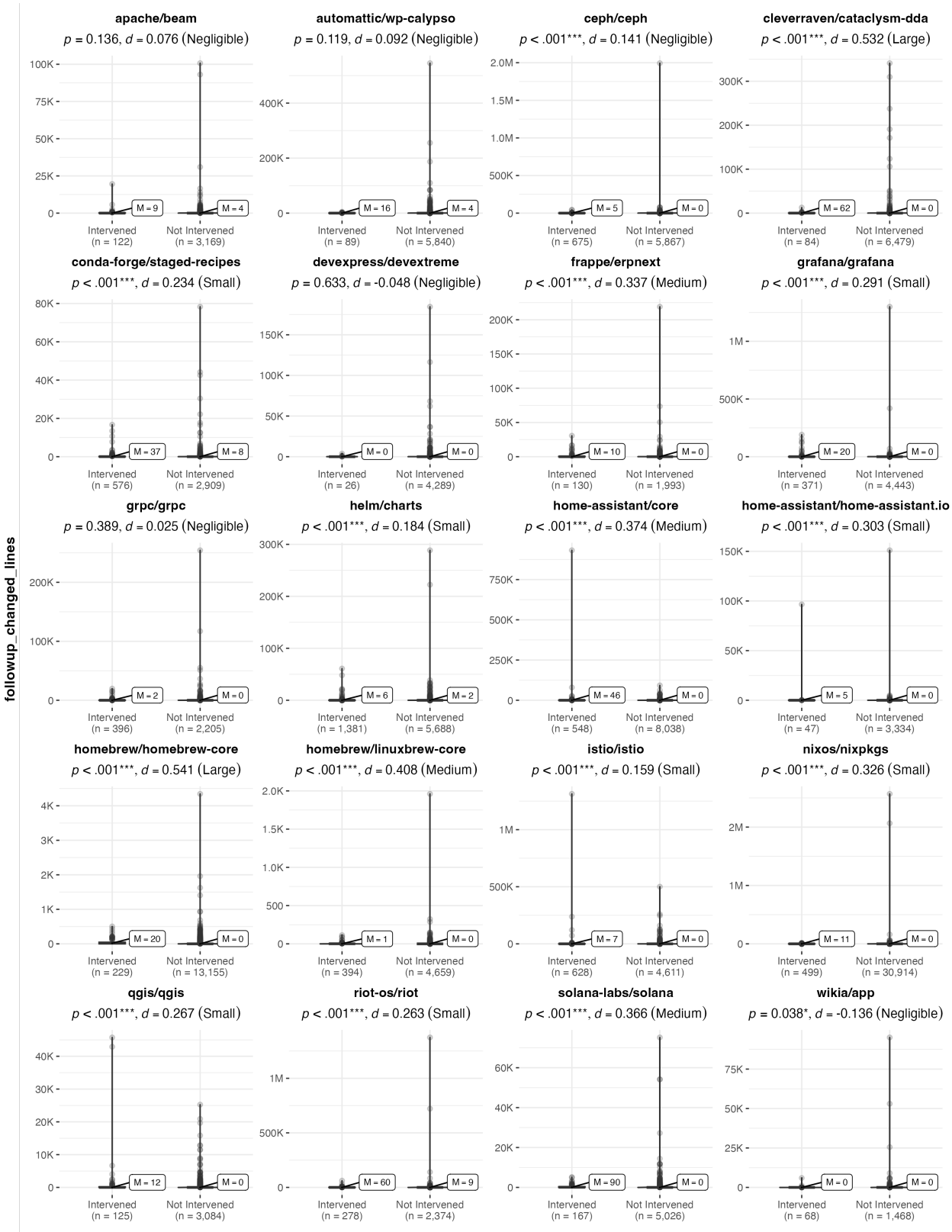
Comparison of intervened and not intervened PRs regarding their number of initial commits across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



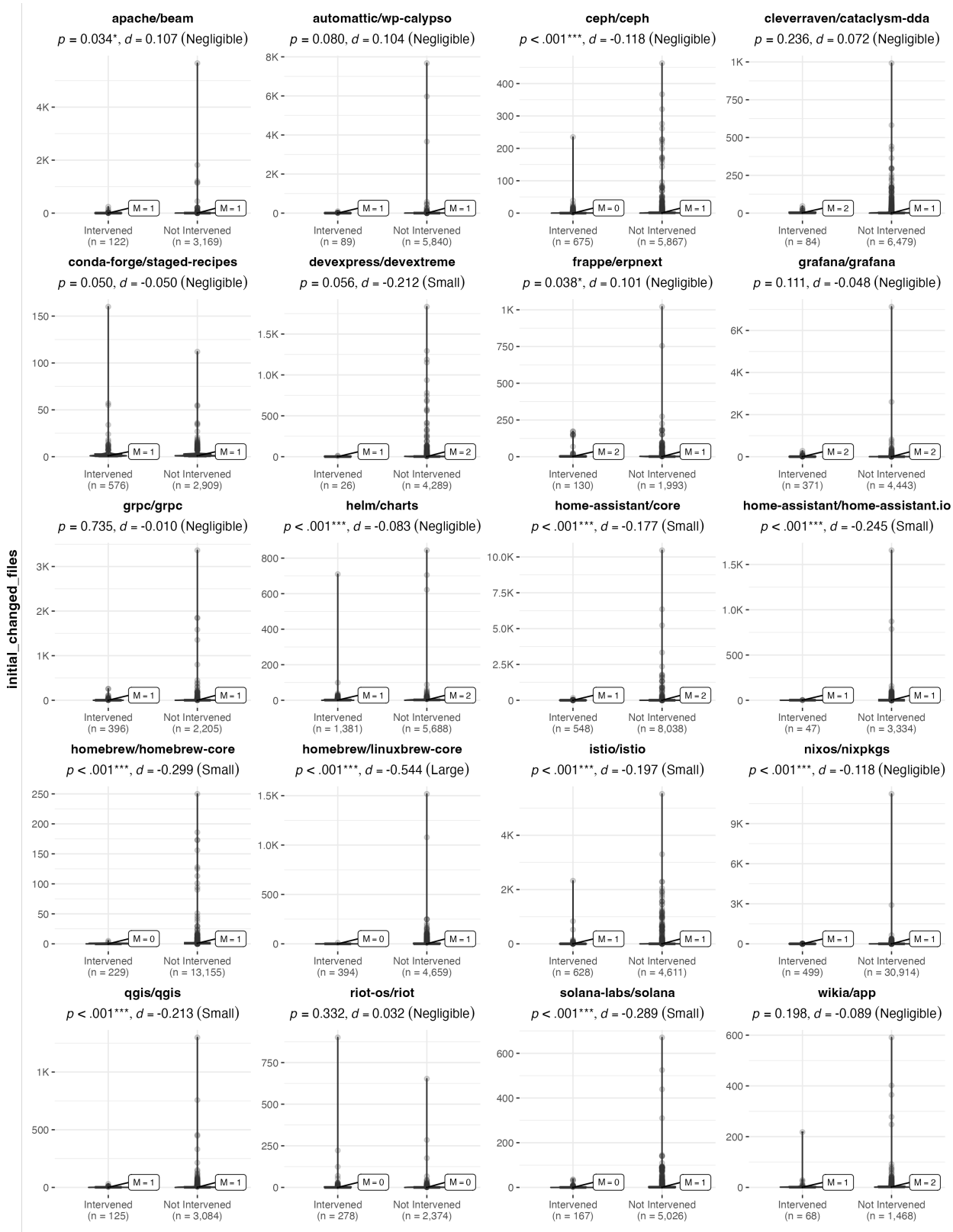
Comparison of intervened and not intervened PRs regarding their number of follow-up commits across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



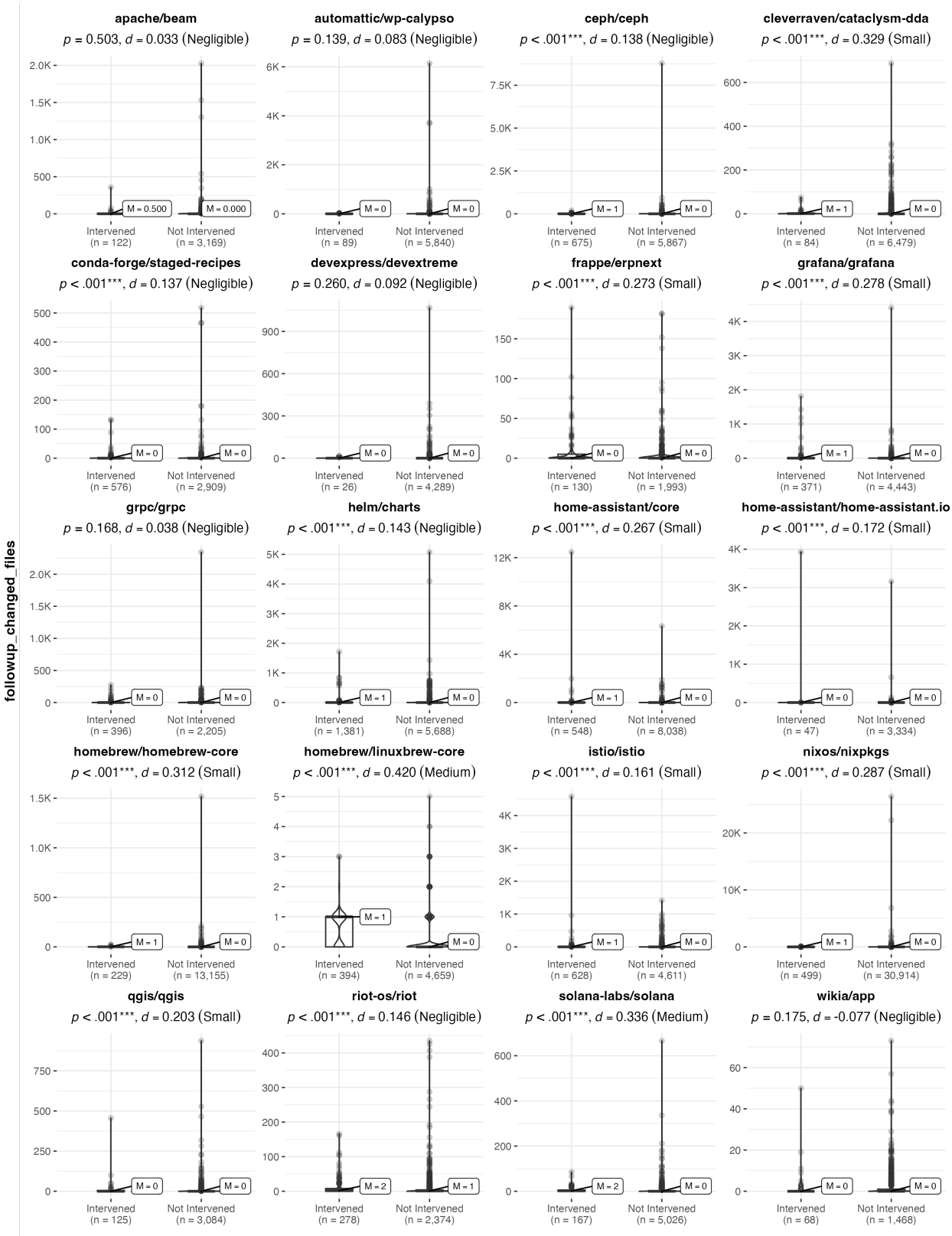
Comparison of intervened and not intervened PRs regarding their number of initial changed lines across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



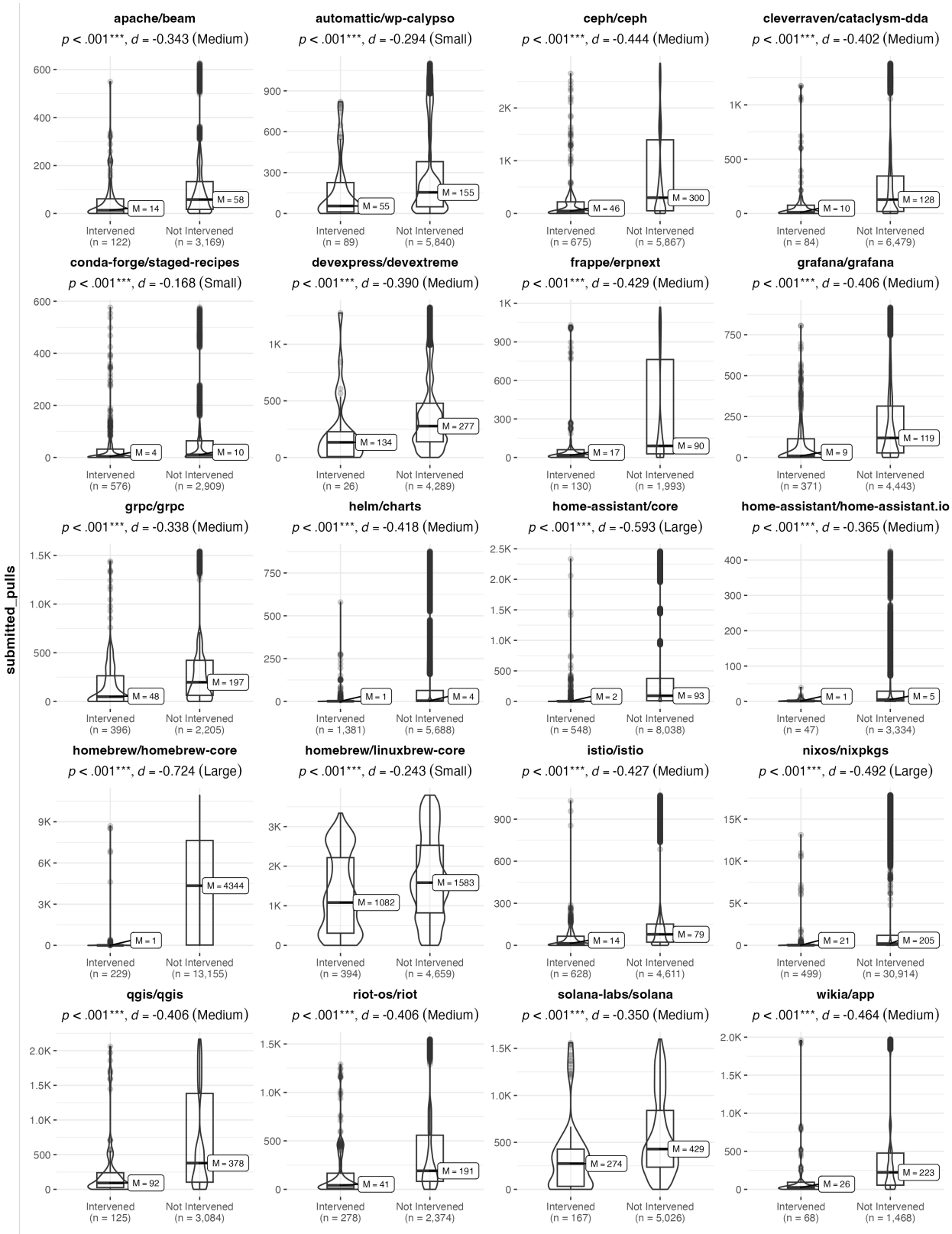
Comparison of intervened and not intervened PRs regarding their number of follow-up changed lines across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



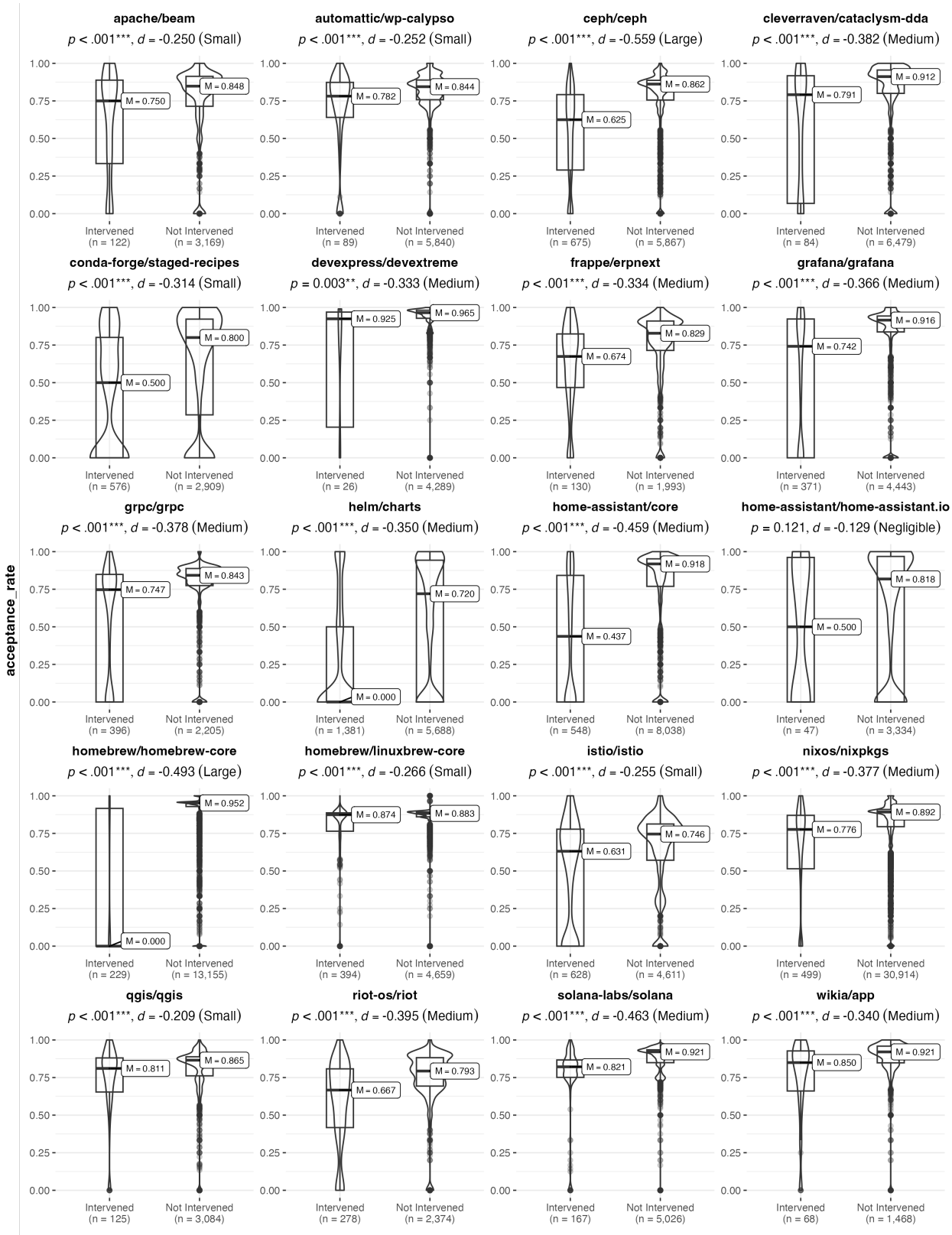
Comparison of intervened and not intervened PRs regarding their number of initial changed files across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



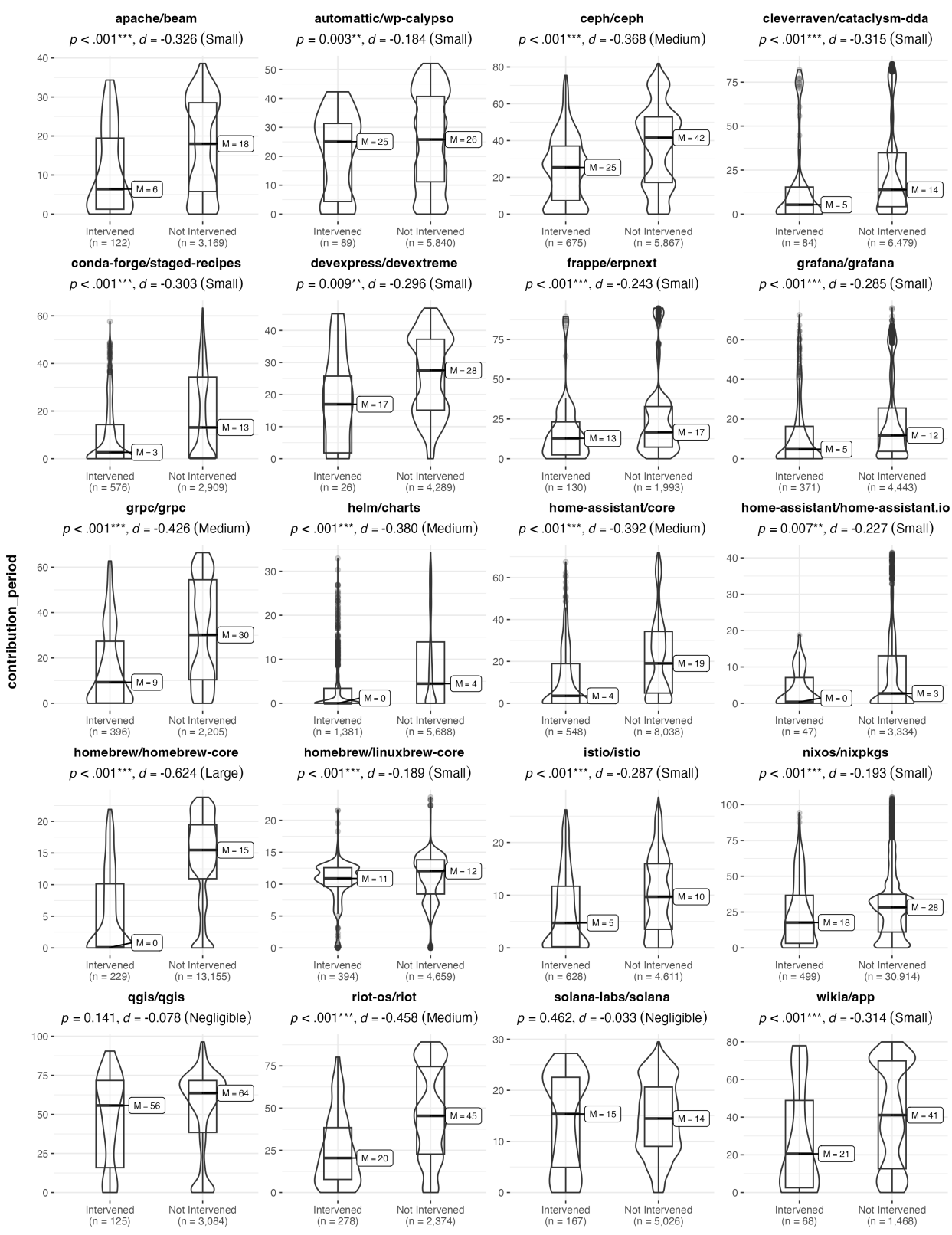
Comparison of intervened and not intervened PRs regarding their number of follow-up changed files across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



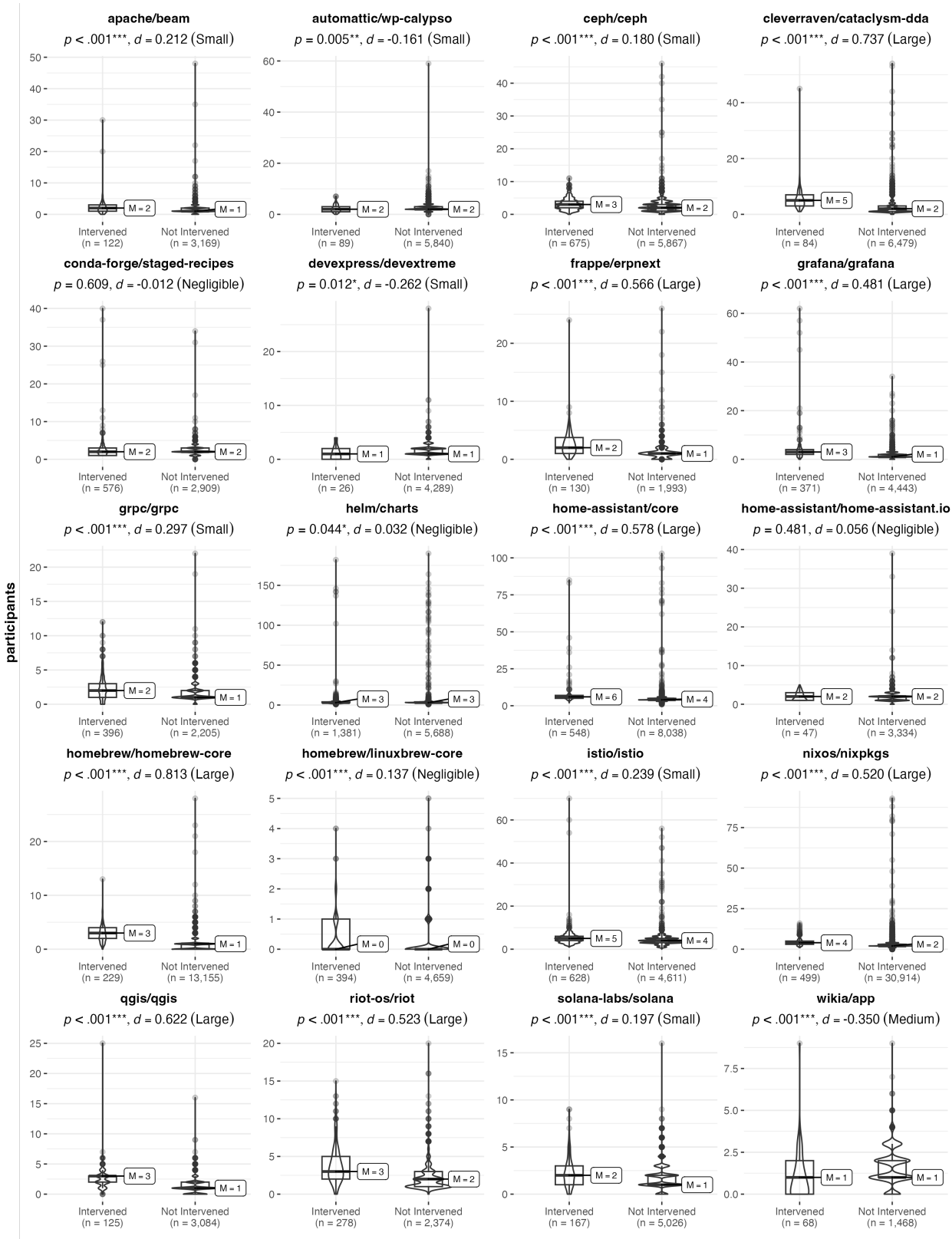
Comparison of intervened and not intervened PRs regarding the number of prior PRs by their contributors across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



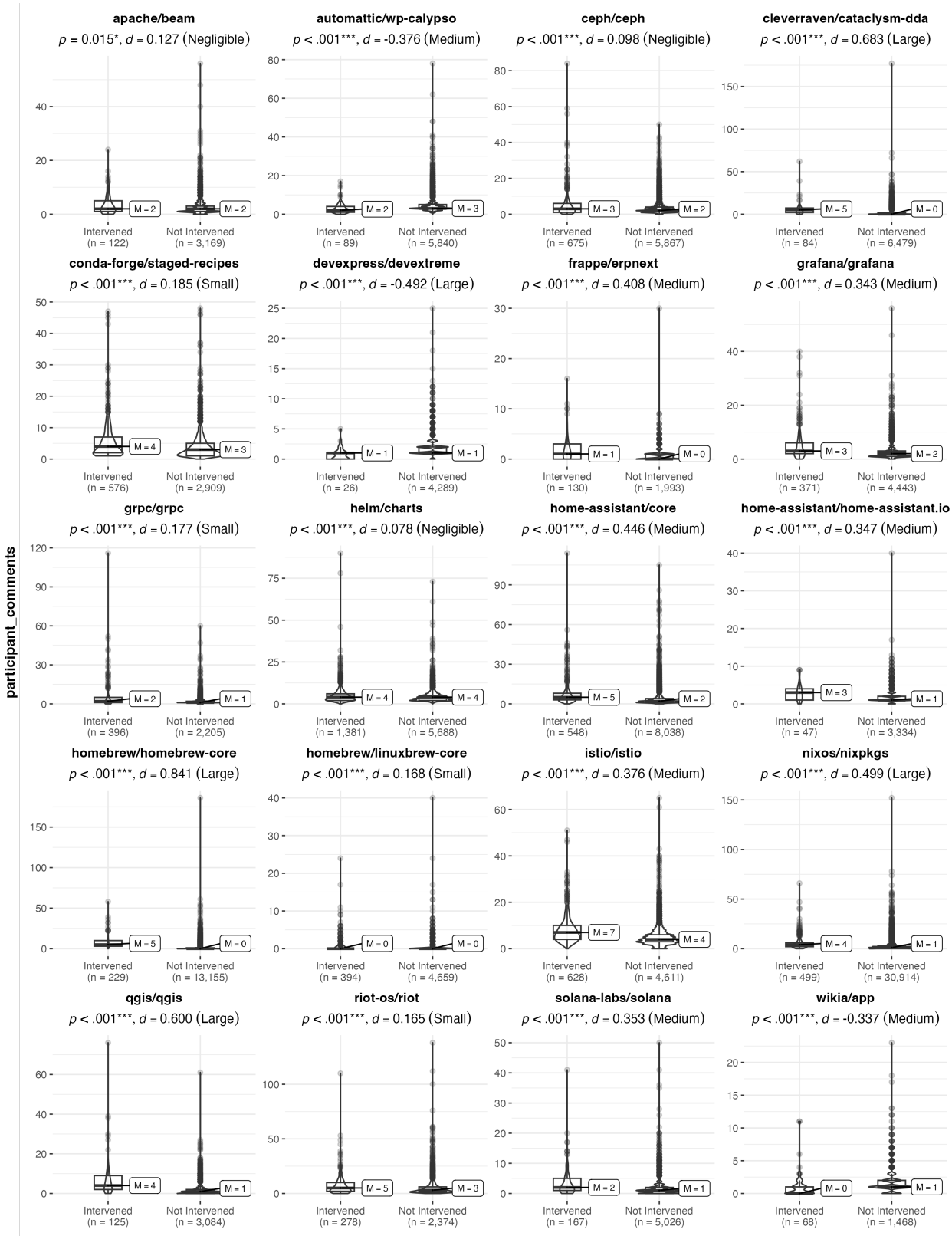
Comparison of intervened and not intervened PRs regarding the acceptance rate of their contributors across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



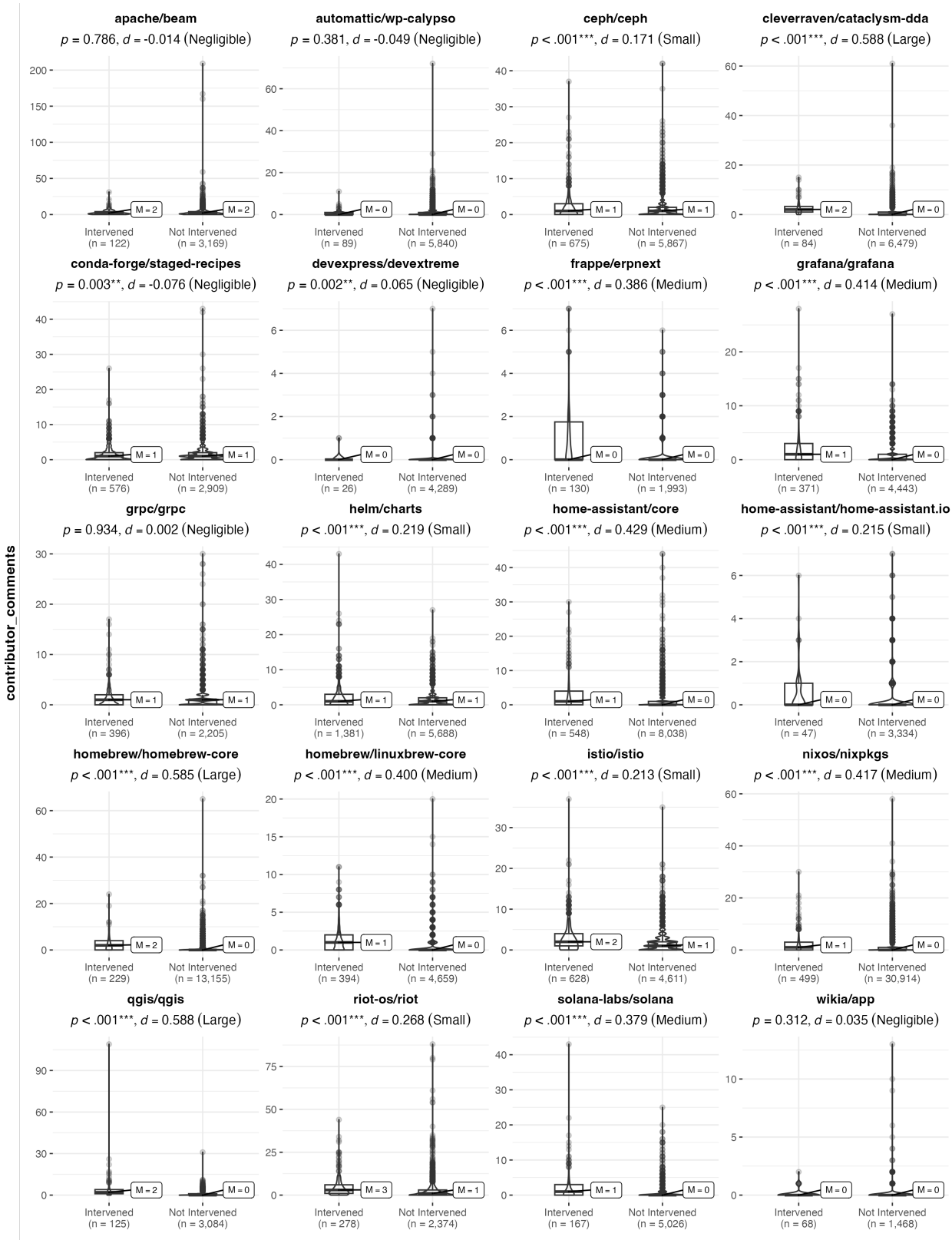
Comparison of intervened and not intervened PRs regarding the contribution period of their contributors across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



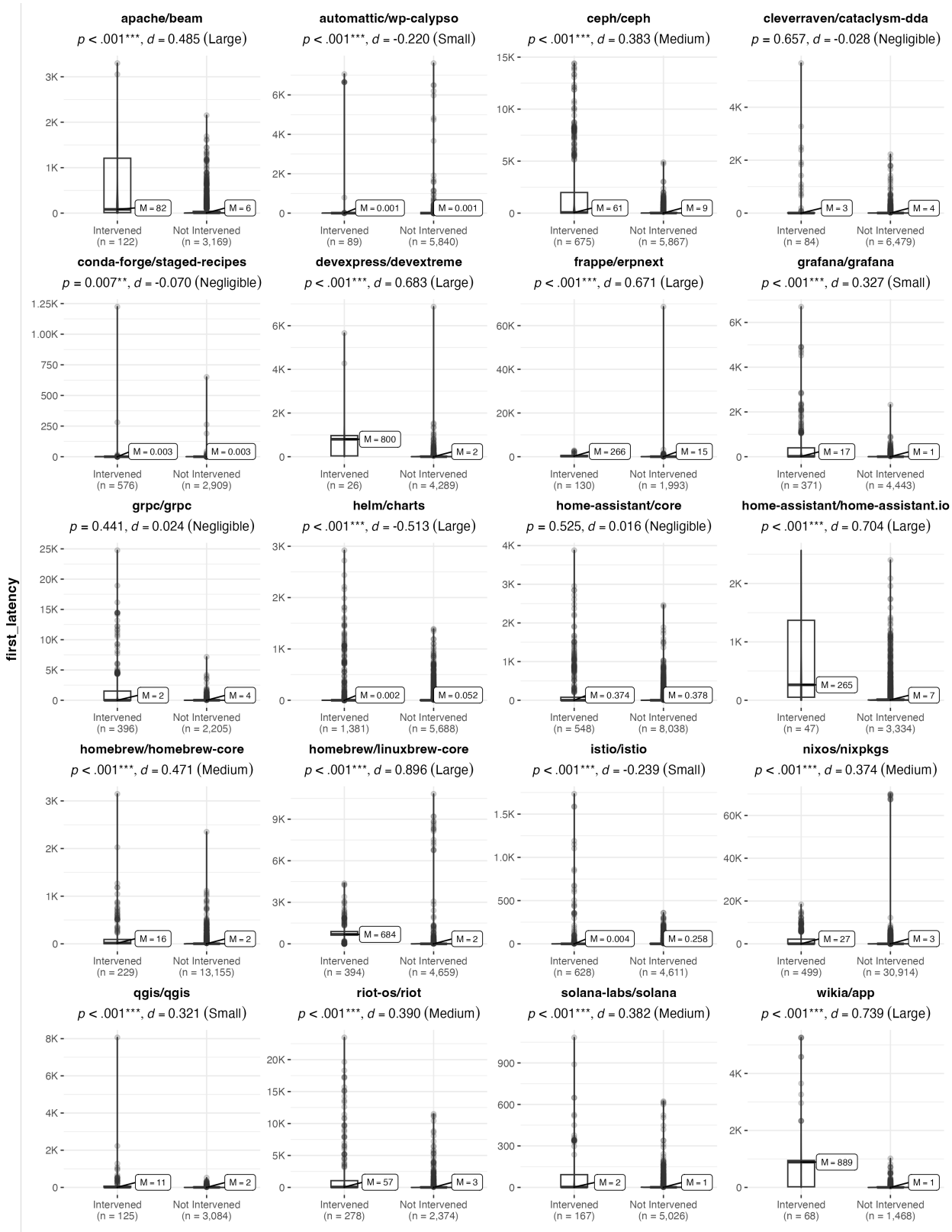
Comparison of intervened and not intervened PRs regarding the number of participants in their review process across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



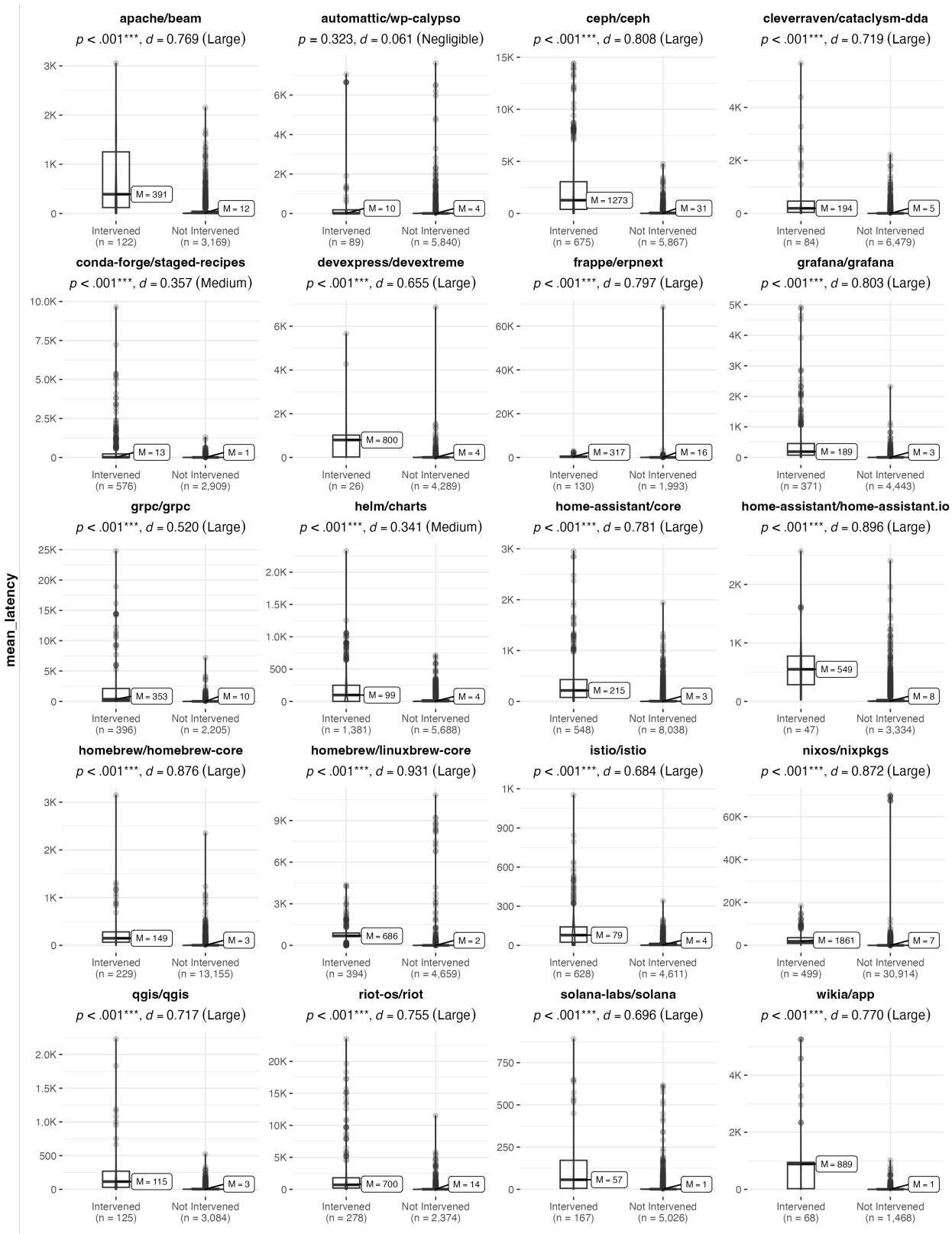
Comparison of intervened and not intervened PRs regarding the number of participant comments in their review process across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



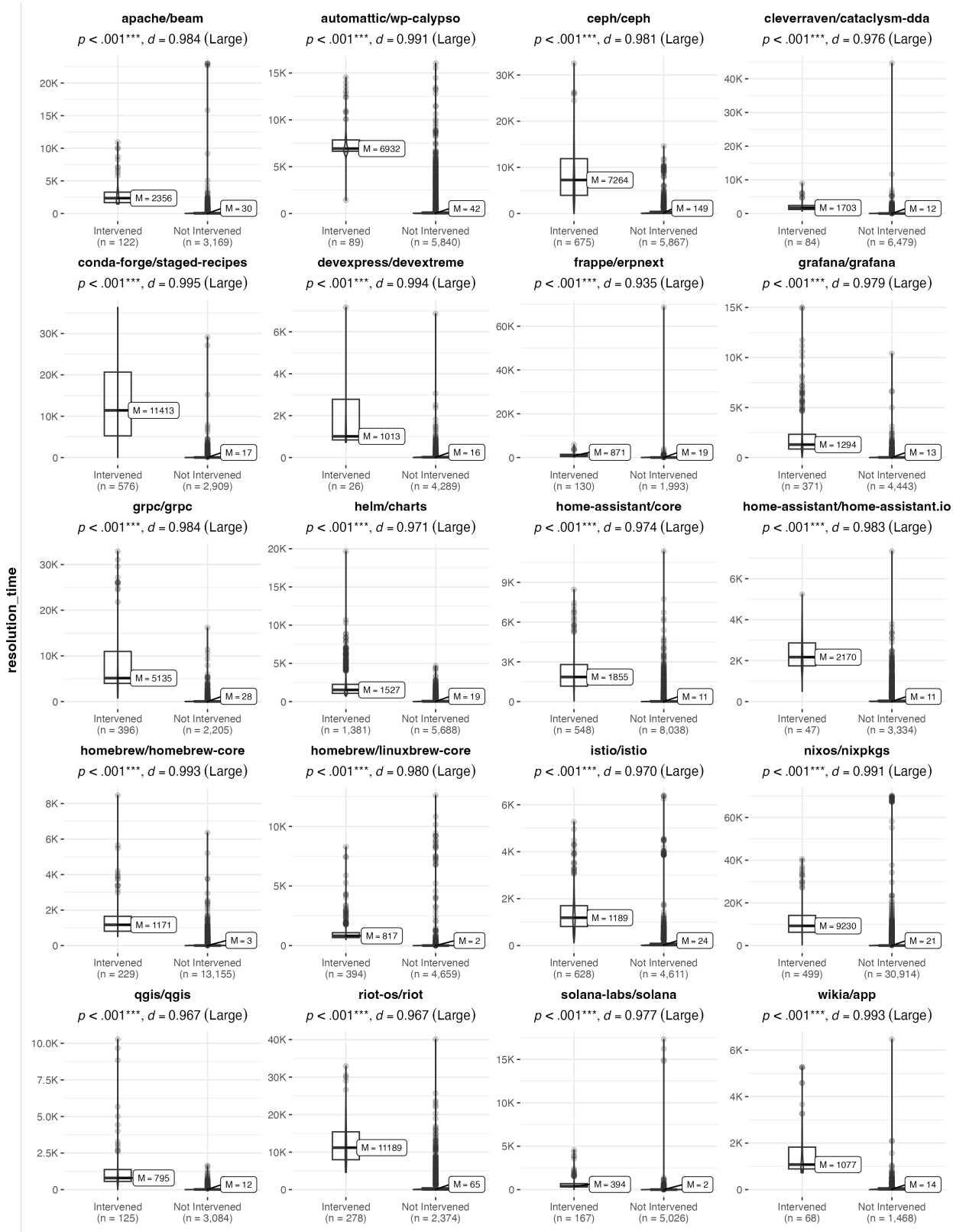
Comparison of intervened and not intervened PRs regarding the number of contributor comments in their review process across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$



Comparison of intervened and not intervened PRs regarding their first response latency across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

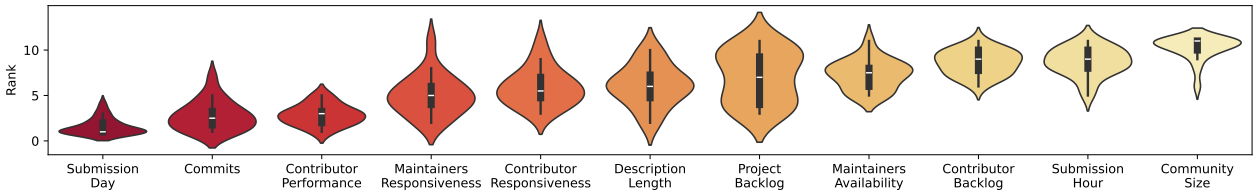


Comparison of intervened and not intervened PRs regarding their mean response latency across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

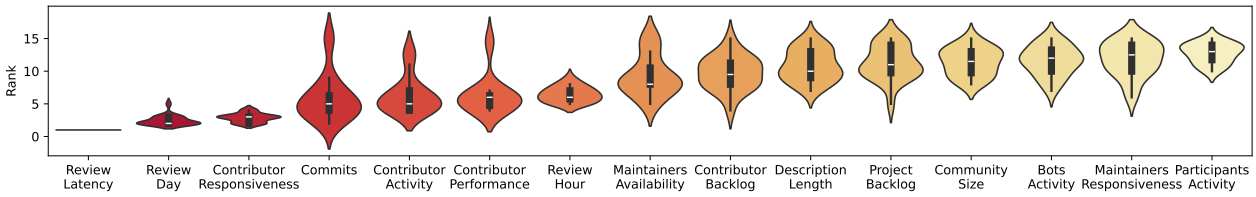


Comparison of intervened and not intervened PRs regarding their resolution time across the studied projects. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Cross-Project Feature Importance



Ranking of the importance of different features for predicting the first response latency of maintainers in a cross-project scenario.



Ranking of the importance of different features for predicting the first response latency of contributors in a cross-project scenario.