Bachelor's Thesis

An Exploratory Study of Bot Activity and Its Effects on Open-Source Software Projects

Côme Lance-Perlick

September 3, 2025

Advisor:
Dr. Thomas Bock Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering
Prof. Dr. Andreas Zeller CISPA

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

Erklärung Statement

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne die Beteiligung dritter Personen verfasst habe, und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht. Insbesondere bestätige ich hiermit, dass ich bei der Erstellung der nachfolgenden Arbeit mittels künstlicher Intelligenz betriebene Software (z. B. ChatGPT) ausschließlich zur Textüberarbeitung/-korrektur und zur Code-Vervollständigung und nicht zur Bearbeitung der in der Arbeit aufgeworfenen Fragestellungen zu Hilfe genommen habe. Alle mittels künstlicher Intelligenz betriebenen Software (z. B. ChatGPT) generierten und/oder bearbeiteten Teile der Arbeit wurden kenntlich gemacht und als Hilfsmittel angegeben. Ich erkläre mich damit einverstanden, dass die Arbeit mittels eines Plagiatsprogrammes auf die Nutzung einer solchen Software überprüft wird. Mir ist bewusst, dass der Verstoß gegen diese Versicherung zum Nichtbestehen der Prüfung bis hin zum Verlust des Prüfungsanspruchs führen kann.

I hereby declare that I have written this thesis independently and without the involvement of third parties, and that I have used no sources or aids other than those indicated. All passages taken directly or indirectly from publications or other external sources have been identified as such. In particular, I confirm that I have used AI-based software (e.g., ChatGPT) exclusively for the following permitted sub-tasks: text rewriting/revision and code completion, and not to address or formulate the main research questions of the thesis. All parts of the thesis that were generated and/or edited using AI-based software (e.g., ChatGPT) have been disclosed and documented in accordance with the documentation requirements. I agree that the thesis may be checked using plagiarism detection software, including checks for the use of such software. I am aware that any violation of this declaration may result in failing the examination and lead to losing the right to be examined.

Saarbrücken, _____ _____
(Datum Date) (Unterschrift Signature)

Einverständniserklärung (optional) Declaration of Consent (optional)

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____
(Datum Date) (Unterschrift Signature)

Abstract

Bots are increasingly used in open-source software (OSS) projects to support developers across various activities, yet their impact on project dynamics remains unclear. While they automate tasks, they may also reshape developer networks in ways that obscure team structures and workflows. To address this, we classify bots based on their activity and analyze how they integrate into developer networks across different project contexts. We study changes in network structure and key project metrics, such as issue resolution rates and pull request handling times, following bot adoption.

We interpret and explain differences in bot activity across projects, finding that most bots exhibit short or very light activity, and their activity rates are rarely linked to indicators like the number of developers. We find that bots consistently occupy high positions in network centrality, particularly in larger projects, even with minimal participation. Their hierarchy centrality is strongly influenced by the types of issue events they trigger. High-activity bots often achieve centrality ranks above core developers, with bots focused on commenting most likely to dominate both in hierarchy and eigenvector centrality. Additionally, high-activity bots produce more consistent shifts in project metrics , particularly those that add or modify commits. Bots triggering labeling or state updates also measurably impact issue turnover, though most bots exhibit limited influence overall.

Overall, our findings point to complex interactions between bot behavior, project structure, and usage context, highlighting the need for further exploration of their role in shaping OSS collaboration.

Contents

List of Figures

List of Tables

1 Introduction

Among their different tasks, software developers may have to deal with many tedious or repetitive endeavors: code reviews, mentor newcomers, triage bugs, maintenance, etc.. [21] To improve effectiveness and productivity, a lot of open-source software (OSS) projects use bots.

Software-engineering bots are programs that can automate [30] or facilitate tasks [39]. They have very different functionalities from one another: A bot like GrammarlyBot[1] will check all README files for grammar mistakes, whilst Dependabot[2] keeps the project's dependencies up-to-date making sure the project can compile. Other bots may just perform tests on the code (like CI bots [2]) or ask new users to sign a document when they first enter a project (CLA assistant[3], JSF CLA Assistant[4]). Through automation, bots can also assure reliability and predictability, and reduce both human errors and stress [2]. And thanks to bots, many tedious tasks can be dealt with, leaving human developers more time to deal with more complex or interesting work.

Whilst the variety of tasks bots can help with makes them a great tool, it is often hard to understand what they do and whether or not they have a beneficial impact on a project. Sometimes, the bots do not perform the right actions or are not used correctly due to them being poorly documented [37, 42]. Additionally, bot interactions during pull requests can frustrate developers, which may cause them to leave negative feedback or even to stop contributing to projects [8, 38]. There have also been signs of "bot skepticism" [9]: Pull requests submitted by bots are less likely to be merged into the project compared to similar ones made by human developers [9]. Some studies found that certain bots would often inundate maintainers with notifications, and that only a third of their pull requests were eventually merged [29, 38]. All of this leads to certain bots being added for only short periods of time or to being discontinued after trial periods. Hence the bots vary not only in their functionalities, but also in how they are employed in a project.

Prior work has shown that modeling developers' collaborations and communications can be used to understand the structure, evolution, and role differentiation within OSS projects [6]. In particular, Bock et al. [6] show that constructing developer networks (in which collaborative and communicative interactions are mapped) can be used to distinguish both developer roles. For example, core developers (key contributors in OSS projects) often emerge as central figures within these networks [6]. Building on top of this, we can extend

1 https://github.com/apps/grammarly-bot/
2 https://github.com/dependabot/
3 https://github.com/cla-assistant/
4 https://github.com/jsf-clabot/

the analysis to include bots. By positioning bots within the same network framework, we may assess their importance and determine how they influence team collaboration.

To achieve this, we study bot activity across 10 OSS projects. First, we identify the bots and categorize them into activity roles based on their duration of activity, and the types of issue-related events they perform. Next, we construct developer collaboration networks for each project. This allows us to investigate whether, and how, the integration of bots—depending on their activity roles—affects developer interactions. Finally, we examine more general project-level outcomes potentially influenced by bot adoption. Specifically, we compare metrics such as issue resolution rates, the number of monthly pull requests, and pull request handling times before and after bot introduction, using a one-year window on each side.

Through this exploratory analysis, our goal is to better understand how bots affect open source projects, both in terms of developer collaboration and overall project performance.

1.1 Overview

In Chapter 2, we introduce key elements such as developer networks, centrality measures, roles in developer networks and GitHub issue events relevant to our study, while also presenting prior work on bots in OSS projects, covering their definitions, detection methods, activity patterns, and their measurable impact on project outcomes.

In Chapter 3, we present our research questions and describe our approach for studying the roles and impacts of bots in GitHub projects: we explain how we identify and classify bots based on their activity, how we analyze their positions within developer networks using centrality measures, and how we plan to assess their effects on project metrics through before–after adoption comparisons and statistical testing.

In Chapter 4, we present the results of our exploratory study, structured around the three research questions. For each question, we report the findings of our analyses followed by a discussion that interprets these results in context.

In Chapter 5, we discuss the internal and external threats to the validity of our results.

In Chapter 6, we present the conclusion of this thesis and provide an outlook on future work.

2

Background and Related Work

This section provides an overview of the main ideas on which we build for this thesis. Research on software bots in open-source software (OSS) covers several themes: defining and classifying bots, identifying bots in repositories, characterizing bot activities, understanding human–bot interactions in issues and pull requests, analyzing how bots fit into developer networks, and measuring the effects of bot adoption on project outcomes.

We begin by discussing what software bots are in the context of software engineering. In doing so, we examine how the term bot is interpreted differently by researchers and practitioners, with several studies showing that a single definition fails to capture the range of perceptions and uses in practice. We also review how previous work has approached the classification of bots. Next, we outline the main techniques for detecting bots, including rule-based, pattern-based, and machine learning approaches. We then introduce the concept of developer roles in OSS, particularly the distinction between core developers—those who contribute regularly—and peripheral developers, who participate less frequently. Building on this, we explore how developer networks are constructed, typically from interactions in code contributions, mailing lists, and issue trackers.

We also present key network analysis metrics that are used to quantify the structure and dynamics of developer communities. These metrics help us understand both human and bot participation in OSS projects and reveal broader trends in collaboration and influence over time. Finally, we present studies that have measured the effects of bot adoption on project dynamics and give an overview of the GitHub issue events that are the most relevant to this thesis.

2.1 Bots – Definition, Perception, and Taxonomy

Numbers of papers have been trying to define what bots in the context of software engineering are [32, 34, 35, 39]. What the papers agree on is that the purpose of bots is improving effectiveness and productivity of developers by facilitating certain tasks. Mainly, they try to improve communication (by mentioning users automatically, or reminding them of events), assist with code quality (code reviews, code smell analysis), manage project workflows, or track issues [32, 34, 35, 39].

Santhanam et al. [32] explain that software bots "come in diverse forms" and have very varied functionalities. Inspired by a talk by Sven Peters[1], they have decided to categorize bots in software engineering through systematic mapping to understand the current landscape, explore the reasons and methods behind bot usage in software engineering, and identify

1 https://svenpet.com/talks/rise-of-the-machines-automate-your-development/

unexplored research areas and potential applications for bots. They have first gathered pre-existing bot classifications (that often overlap one another) such as Storey and Zagalskys' [34]: Storey and Zagalsky identified bots as 'Codebots' when they helped make coding more efficient and effective for developers (including all bots that contribute towards improving coding tasks), 'Devop bots' when they are used to speed up code deployment/manage development infrastructure and operations, and 'Test bots' when they help in the testing environment [34].

Santhanam et al. [32] explain how bots are applied in multiple different areas, mainly: "managing software repositories", "controlling the software development process (SDP)", and in "question answering systems (Q&A bots)". As this thesis focuses on bots in software repositories, it is the classes in the "managing software repositories" area and specifically those used on GitHub, we are interested in.

To illustrate the functionalities and diversity among bots, here is a list of several bot classes from this area, that we extracted from Santhanam et al. [32], along with examples for each class that we provide.

Dependency Management Bots

- **Examples**: Dependabot[1], Renovate[2]
- **Usage**: Automatically update dependencies and manage security updates.

1 https://github.com/dependabot/
2 https://github.com/renovatebot/

Automated Code Review Bots

- **Examples**: SonarQube[1], Codacy[2]
- **Usage**: Perform code smell checking, test coverage analysis, and style checking.

1 Has become a Github app: https://github.com/apps/sonarqubecloud/
2 https://github.com/codacy/

Continuous Integration Failure Reporting Bots

- **Examples**: CircleCI[1], Travis CI[2]
- **Usage**: Notify developers of CI failures, often integrated with build systems.

1 https://github.com/circleci/
2 https://github.com/travis-ci/

Pull Request Merging Bots

- **Example**: Mergify[1]
- **Usage**: Automatically merge pull requests based on pre-set criteria.

1 https://github.com/Mergifyio/

Automated Software Build Bots

- **Examples**: GitHub Actions[1]
- **Usage**: Manage and automate the software build process.

1 Has been integrated into GitHub (https://docs.github.com/en/actions/)

Stalebots

- **Usage**: Identify and manage stale issues and pull requests in large projects.
- **Note**: Stalebots were widely used but are very controversial; many developers now advise against their usage [27].

License Agreement Signing Bots

- **Example**: CLA Assistant[1]
- **Usage**: Ensure contributors sign license agreements.

1 https://github.com/cla-assistant/

Vulnerability Fixing Bots

- **Example**: Dependabot[1] (also manages security updates)
- **Usage**: Identify and fix security vulnerabilities.

1 https://github.com/dependabot/

Code Reviewer Assignment Bots

- **Examples**: Mention-bot[1], Sankie[2]
- **Usage**: Assign reviewers to pull requests based on context.

1 https://github.com/facebookarchive/mention-bot/
2 Has become a GitHub App: https://github.com/apps/sankiebot/

Automated Testing Bots

- **Examples**: CircleCI, Travis CI
- **Usage**: Run automated tests on new code commits.

Refactoring and Code Cleanup Bots

- **Example**: Refactoring bot[1]
- **Usage**: Perform code refactoring and fix code smells.

1 https://github.com/Refactoring-Bot/

Documentation Bots

- **Example**: GitHub Actions (when integrating to it tools that used to be bots, e.g., Coveralls[1], Codecov[2])
- **Usage**: Generate and update documentation.

1 https://github.com/coverallsapp/github-action/
2 https://github.com/codecov/codecov-action/

Erlenhov et al. [18] take a perception-oriented approach rather than giving a single formal definition of DevBots. Through practitioner interviews, they explore what makes a "Plain Old Development Tool" feel like a "bot" to developers. They identify three personas based on interviewee expectations:

- **Charlie (Chat-bot persona)**: Charlie sees bots as any tool that communicates via natural language, regardless of the functionalities it provides. For Charlie, the "bot" is the conversational interface, not really the automation itself. An example of bots that correspond well to this are bots for ChatOps[2].
- **Alex (Autonomous-bot persona)**: Defines bots by their autonomy—tools that independently monitor and act on simple tasks (like closing stale issues) without human prompts. Tools like Dependabot illustrate Alex's view, as they trigger based on events rather than manual commands.
- **Sam (Smart-bot persona)**: Associates bots with technical sophistication—adaptive, ML or analysis-driven agents capable of interpreting code and making intelligent decisions, beyond simple scripted actions.

Erlenhov et al. [18] emphasize that these personas often overlap and that developers' definitions of "bot" vary with their own experiences and tool use. Their work shows how practitioners think about bots, instead of defining what a bot should be.

Meanwhile, the BOTse seminar [34] gathered different researchers and practitioners together to discuss the present and future of bots in software engineering.

2 https://www.pagerduty.com/blog/what-is-chatops/

Building on perception studies, Wessel et al. [41] analyzed 351 popular GitHub repositories and formulated a functional taxonomy of bots. They describe 13 categories: from continuous-integration (CI) bots and reviewer assigners to license-agreement signing bots and bots that welcome newcomers. They observe that over 30% of OSS projects adopt at least one bot, especially after 2013.

2.2 Bot Identification Methods

Whilst GitHub labels some of the bots, a lot of them are not and need to be detected differently. The works of Golzadeh et al. contribute significantly to software engineering research, not only for bot-focused research but also for any work that studies developer dynamics in OSS projects [1, 5, 6, 20]. Golzadeh et al. [20] developed BoDeGHa, a machine learning-based tool designed to detect bots in GitHub issue and pull request (PR) comments, essentially enabling bot detection across large numbers of projects. Additionally, they provide future researchers with a "Ground-truth dataset" containing 527 GitHub account they have labeled as bots, serving as a foundation for training and evaluating bot detection tools.

Abdellatif et al. [1] build on Golzadeh et al. [20] by generalizing bot detection outside of comments on issues and pull requests. Their tool BotHunter was trained using different machine learning classifiers, and looks at different features they extract directly from GitHub profiles and activity metrics. With their approach, they claim to outperform BoDeGHa and other bot detection tools in recall and precision, at the cost of higher GitHub API usage and longer execution times.

Later, Chidambaram et al. [11] observed that both comment-based and commit-based approaches overlook many other bot activities. They first created a new ground-truth dataset of 2 150 contributors (1 035 bots and 1 115 humans) that includes a wide range of activities performed by both bots and humans. They then extracted up to 300 recent GitHub Events per contributor and mapped these low-level events to 24 activity types [11]. And from each activity type they were able to compute different candidate features. Using a 10-fold grid-search cross-validation over seven classifier families, they found that a Gradient Boosting model achieved the best trade-off between precision and recall [12], where previous models were using Random Forest classifiers instead [1, 12, 20]. This model, called BIMBAS ("Bot Identification Model Based on Activity Sequences"), attains comparable performances to BotHunter but without relying on name-based heuristics (Github account login names, usernames, or display names) [12]. To enable the usage of BIMBAS, Chidambaram et al. implemented RABBIT ("an Activity-Based Bot Identification Tool") as an open source-command-line utility tool. They then tested [12] different bot identification tools on a 860-contributor test set: RABBIT classifies all users in 22 minutes, downloads only 112 MB of data, and issues 2 426 API queries—an order of magnitude more efficient than BoDeGHa (3.83 GB of data and 7.7h of runtime) and BotHunter (0.261 GB of data and around 21h runtime)—while maintaining the same prediction quality both in precision and recall [12].

Wyrich et al. [44] opt for an alternative approach to bot detection that does not involve machine learning models. They combine three methods. First, they consider bot usernames. They assemble a list made up of users officially labeled as bots by the GitHub API (with a

[*bot*] suffix) and users detected using the expression-matching approach of Dey et al. [16], which uses the filter rules: "1) '%bot' and '%robot', 2) '%-bot-%' and '%-robot-%', 3) 'bot-%' and 'robot-%'". They state that this username pattern is widespread among bots, though it risks including a few humans too. Second, they manually review GitHub profiles to identify other criteria indicating that a user is a bot — such as profile descriptions explicitly stating so, or users with over 10,000 contributions per year. Finally, they include the dataset of known bots from Dey et al. [16] to expand their list. Using this combined method, they identify 4,654 distinct bot usernames.

Bock et al. [6] use BoDeGHa and Wyrich et al. [44]'s approach as a means to filter out the bots and explore developer dynamics. They find that BoDeGHa has some limitations (e.g. bots that do not post comments can not be recognized, or humans that use predefined comment templates can be identified as bots), and thus removed some bots manually and complemented the list provided by BoDeGHa by making use of Wyrich et al. [44]'s solution.

2.3 Developer Roles

In software engineering projects, especially in open-source software (OSS), developers take on different roles based on their level of contribution and engagement. Several models have been made to categorize these roles. One of the most well-known is the "onion model" by Nakakoji et al. [31], it describes eight different roles in an OSS community. Roles like passive users, readers, and bug reporters, do not involve direct contributions to the source code. Other roles, such as bug fixers, peripheral developers, active developers, core members, and project leaders, play a more direct part in development [6, 31].

While these models provide a detailed view of developer roles, many researchers [6, 14, 36] simplify them into just two main categories: core developers and peripheral developers. This distinction helps in analyzing developer activity and contributions more effectively. Core developers, sometimes called maintainers [14, 25], contribute regularly to the project. They are often responsible for reviewing code, merging pull requests, and making key decisions about the software. Project leaders, who are usually the initiators of a project, are often grouped with core developers due to their continuous involvement. On the other hand, peripheral developers are less active and their changes are usually smaller [14, 25]. This group includes bug fixers, minor contributors, and sometimes one-time contributors who provide a single patch or bug fix.

Some researchers have also tried to automatically sort developers into core and peripheral groups [6]. They do this in different ways. One method is to check the permissions of a developer—what tasks they are allowed to do—which can show if they are more involved with the project. Other methods count how many contributions or comments a developer makes. Some even use network-based methods, where they look at how developers interact with each other. These approaches help to figure out who works on the project regularly (core developers) and who only helps out once in a while (peripheral developers). Bock et al. [6] identified what network construction-method and which metrics are most accurate at identifying whether or not the role of a developer is core or peripheral. In this thesis, we will use their role detection methods to gather information on the different projects we are studying.

2.4 Developer Networks

Bock's thesis [5] explains how developer networks can be built and how they can be used to assess developer roles and understand the structure of projects. Developer networks are a way to map out how people interact in open-source software projects. They are built from different types of data like co-change networks (which show how developers work on the same pieces of code) [19, 26], mail networks (which capture communication on mailing lists) [4], and issue networks (which trace interactions in issue trackers) [13, 24]. One key part of these networks is centrality, which tells us how connected and important a person is within the project [22]. Among the many network centrality measures available [22], Bock et al. opted for three that are well recognized in the literature and that highlight different facets of centrality [5]:

- Degree Centrality: This measures the number of direct connections a developer has. A higher score means more direct interactions [6, 7, 33].
- Eigenvector Centrality: This metric looks at both direct connections and how well-connected a developer's contacts are. A developer with high eigenvector centrality is linked to other influential members of the network [6, 7, 33].
- Hierarchy Centrality: This helps show the layered structure in the project by indicating who holds more influence or decision-making power [6, 25, 26]. It combines both node degree centrality and the clustering coefficient into a score. Being at the top of the hierarchy (indicated by a high hierarchy centrality) implies high degree centrality and low clustering coefficient. It often implies having a good overview of the project, being able to supervise it and branching out to many different clusters.

These metrics are not only useful for understanding the roles of human developers but can also be applied to bots. For example, by comparing the centrality scores of a bot to those of human developers, we can see how much the bot is involved in daily communications and coordination, much like how researchers compare human developers among each other [15, 17]. If a bot has high eigenvector centrality, it means the bot is connected to other key players, showing that it has a significant impact on project management tasks like issue tracking or code review.

Developer networks also allow us to spot emerging patterns in OSS projects [5]. Over time, they can reveal shifts in collaboration, the rise of new influential groups, or even changes in how bots interact with human contributors [5]. These patterns provide valuable insights into the evolving structure of a project and help us understand where responsibilities and influence are moving.

In short, using metrics like degree, eigenvector, and hierarchy centrality, researchers can effectively measure the impact of any developer within a project and compare its role to that of other developers [5, 17, 25], while also uncovering trends that indicate how the project is developing over time [5, 15, 26]. Since this approach applies to any developer, it can just as well be used to understand the role and impact of bot developers — something we will explore in this thesis.

2.5 Measuring Bot Adoption Effects

Several studies have explored how bot adoption changes project dynamics. These studies [10, 28, 40, 43, 45] typically use Regression Discontinuity Design (RDD) to estimate changes in project metrics before and after a bot is introduced, relying on large-scale datasets with hundreds to thousands of projects. Wessel et al. [40] explain the main idea behind RDD: "The technique is based on the assumption that if the intervention does not affect the outcome, there would be no discontinuity, and the outcome would be continuous over time".

Many of these studies group the bots by type, with Wessel et al. [41] relying on the classification they established in their previous study. Zhao et al. [45] were the first to use RDD to understand the impact of adopting a bot in GitHub projects. Zhao et al. [45] and Cassee et al. [10] both focused on the impact of Travis CI. Both studies on continuous integration (CI) tools found that the number of merged PRs increased, the time to merge or close a PR also increased (possibly due to developers waiting on CI results), and the number of closed issues increased.

Wessel et al. [40] worked on code review bots and, based on Kinsman et al.'s work [28], they wrote another paper about GitHub Actions Bot [43]. In their papers [40, 43], they study project-level metrics one year before and one year after bot adoption. They have considered the following metrics:

- Number of merged/rejected PRs
- Number of comments (excluding bots) on PRs
- Time to merge or close PRs
- Number of commits per PR

Prior to using RDD, they considered an exploratory case study involving only two projects in which they evaluated their results using the Mann-Whitney U test and Cliff's Delta to compare pre- and post-bot adoption.

Kinsman et al. [28] and Wessel et al. [43]'s observation found that adoption of GitHub Actions led to fewer accepted PRs (but more comments in accepted PRs), more rejected PRs (with fewer comments), fewer commits in accepted PRs, more commits in rejected PRs and longer time to close accepted PRs. Contradicting results with their work on code review bots, notably in the rate of PR rejections, can be explained due to fundamental differences in how GitHub Actions and code review bots operate [32]. GitHub Actions support workflows like CI, utilities, deployments, and publishing, while code review bots are dedicated to reviewing code.

2.6 GitHub Issue Events

GitHub provides a detailed event log[3] that records actions performed on issues and pull requests. In this thesis, we rely on these events to characterize the activities of both human

3 GitHub events documentation:
https://docs.github.com/en/webhooks-and-events/events/github-event-types/.

developers and bots. Although GitHub tracks many different event types, we restrict our focus to those that appeared in the projects and bots studied. The most relevant events in our dataset are:

- `created`: Logged when a new PR or issue is first created in the repository.
- `merged`: Signals successful integration of a PR into the main branch.
- `commit_added`: Records when new commits are added to an open PR.
- `head_ref_force_pushed` / `head_ref_deleted`: Signals forced pushes of PRs or branch deletions of PRs
- `assigned` / `unassigned`: Marks when responsibility for an issue or pull request is given to, or removed from, a contributor. Bots may trigger these automatically.
- `labeled` / `unlabeled`: Signals the addition or removal of labels on issues, typically used to categorize or prioritize issues.
- `commented`: Signals a comment added to a discussion around an issue or PR.
- `referenced` / `referenced_by` / `add_link`: These three events indicate cross-linking between issues, PRs, or commits. They highlight dependencies and relationships across project artifacts.
- `mentioned`: Logged when a user or bot is explicitly tagged in an issue or PR discussion.
- `review_requested` / `review_request_removed`: Indicates that a contributor has been asked to review a PR (or that the request was withdrawn). Some bots automate this process.
- `state_updated`: Records updates to the state of an issue or PR (e.g., transitioning from draft to ready for review).
- `locked` / `unlocked`: Restricts or reopens discussion on an issue or PR, often used to enforce repository maintenance policies. Bots like *lockbot*[4] commonly trigger these.
- `subscribed` / `unsubscribed`: Tracks when contributors start or stop following an issue in order to receive notifications.

These events are reliable tools that we will use in this thesis to analyze and comprehend interactions and contributions of any contributor in OSS projects.

4 https://github.com/apps/lock/

3

Methodology

In this chapter, we first introduce and motivate our research questions. We then describe our approach to answering them in the Operationalization section. Finally, we outline our expectations based on preliminary research and discuss potential threats to the validity of our results.

3.1 Research Questions

The goal of this research is to better understand the roles and impact of bots in GitHub projects by categorizing them based on their activity and analyzing how their roles influence developer collaboration and project dynamics.

Bot interactions during pull requests can frustrate developers, which may cause them to leave negative feedback or even to stop contributing to projects [8, 38]. Some studies found that certain bots would often inundate maintainers with notifications, and only a third of the bot-created pull requests were eventually merged [29, 38]. Sometimes, the bots do not perform the right actions or are not used correctly due to them being poorly documented [37, 42]. All of this leads to certain bots being added for short periods of time or being discontinued after trial periods. Hence, bots vary not only in their functionalities, but also in how they are employed in a project. That is why we want to address RQ_1:

RQ_1 What roles do bots take in GitHub projects, and which specific tasks do they perform?

We have noticed that previous work hints at bots being used in widely different ways. Even though bots already have very different functionalities from one another, how and if a project is going to use these functionalities makes evaluating their impact more difficult. This is why, depending on their role, we might note that developer networks can look totally different, once bots have been brought into the project. Previous work by Bock et al. [6] actually filtered out the bots in order to better comprehend project dynamics with their developer networks . That is why we want to answer RQ_2:

RQ_2 How are these bot roles reflected in the structure of the project's developer network?

The presence of bots can potentially distort how networks need to be interpreted. Especially when trying to identify roles of developers. Can bots become more central than

core developers? Will bots dominate the networks, or will core developers still maintain control over the networks? Depending on the numbers of core developers or bots, there may be many different observations possible. This is why we want to consider that different aspects (e.g., bot activity, or number of developers) can vary from project to project or across a project's lifetime. By accounting for the specific roles of bots and the projects they are embedded in, we may be able to identify diverse cases for each role and sometimes compare whether their influence is consistent across projects with similar characteristics; potentially making it easier to comprehend the evolution of OSS projects that involve bots.

Much like the works of Wessel et al.[40, 43], we aim to understand other effects of adopting bots, and relate these to the activity roles we are defining in RQ_1.

RQ_3 What impact do specific activity roles have on the project after bots have joined them? How do different bot activity roles influence project metrics after bot adoption?

Besides looking at how central bots are in developer networks, we can also study their impact by checking project-level metrics, such as how fast pull requests or issues get resolved. Unlike past research that looks at many projects, we focus on fewer cases to study them in more detail. This helps us better understand the influence of bots and supports the way we grouped their roles in RQ1.

3.2 Operationalization

In this section, we present our approach to answering the research questions. First, we explain how we gather data from various projects, construct developer networks, and identify participating bots. We then consider each research question individually to explain our methodology.

3.2.1 Collecting the data

For project selection we rely on a subset of the projects investigated by Bock et al. [6], Bock [5] and Hechtl et al. [23]. The selection amounts to 10 projects that have been active for a long time and have multiple highly active bots. Using the tool GitHubWrapper we extract issue meta-data from GitHub, the data includes which user initiated which event within which issue or PR, and at which time [6]. We use codeface-extraction[1] to merge commit and issue data together and match together developers that use the same name or e-mail address [6].

Bock et al. [6]'s contribution and library coronet[2] can be used to assess developer roles based on the centrality of a developer. It uses that developers with centrality values belong-

1 https://github.com/se-sic/codeface-extraction/
2 https://se-sic.github.io/coronet/

ing to the upper 20% quantile are classed as core, while the remaining ones as peripheral, essentially creating an 80% threshold developers have to be above to be considered core [6]. Coronet also allows to read data extracted with codeface and delivers possibilities of creating different developer networks and get their associated metrics.

To identify bots, we use GitHub's internal [bot] labeling, combined with BoDeGHa and a manually curated subset of the dataset by Wyrich et al. [44]. In addition, we manually exclude accounts falsely identified as bots by BoDeGHa by inspecting their GitHub profiles for human-like characteristics.

We conducted a preliminary investigation during which we identified all active bots within the repositories and analyzed their activity. Additionally, we manually identified some humans that were labeled as bots because of bot-like behavior. This exploration was very helpful to familiarize ourselves with the projects and further establish our methodology.

3.2.2 RQ1 – Bot Activity Roles

The goal of this research question is to qualitatively differentiate bots in a list of activity roles and distinguishing what different events they trigger. We define the activity roles based on three aspects: **Activity duration**, **activity rate** and **activity type**.

Activity duration is measured as the time period from the first to the last issue event a bot triggers. However this needs to be interpreted carefully, as some bots may trigger events sporadically: They can be active for a short period, then become silent, and active again much later. This pattern can give the false impression of continuous activity, virtually increasing the activity duration, though they have only been active during two brief intervals. To remedy this, we check for regular activity and reclassify bots that have had short activity periods scattered at different points in time. The activity duration is ultimately expressed as a number of calendar months. The bots get attributed to one of three groups based on their activity duration: Short activity, medium-lengthed activity and long activity. We determine these groups based on the distribution of all activity rates, with bot durations bellow the first quartile (Q1) of the distribution belonging to the short activity group, bots with durations above the third quartile (Q3) belonging to the long activity group, and bots with duration between Q1 and Q2 belonging to the medium-lengthed group.

We define activity rate as the total number of events bots have triggered divided by their activity duration :

$$activity\_rate = \frac{total\_activity}{activity\_duration}$$

Similarly to activity duration, the bots are attributed to one of three activity-rate groups based on the overall distribution of this metric: Light activity rate (bellow Q1), intense activity rate (above Q3), and medium activity rate (between Q1 and Q3).

During our preliminary investigations (see Subsection 3.2.1) of the bots across the projects, we also noticed that a lot of bots exhibited little to no activity, being triggered only once or twice or simply mentioned in the project without actually participating in it. These bots were typically either tested out by maintainers for a short trial period or triggered "referenced", "referenced_by" or "mentioned" events during their activity in other repositories (see section 2.6). We label these bots as "Trial bots" and exclude them from our analysis when computing

distributions and bot adoption effects, as their minimal activity likely had long-lasting impact on the projects.

The third aspect, activity type, is attributed to the bots based on what issue events they have triggered. To determine the activity types of the bots, we classify them based on the types of issue events they trigger. This approach is well-suited because GitHub issue events are closely tied to specific bot capabilities and functions. For example, a bot like Lockbot is limited to locking and unlocking actions, while labeling bots (e.g., label-actionsbot) focus on managing GitHub issue labels and opening or closing issues. By grouping bots that trigger similar events, we define distinct activity categories that reflect how they interact with other developers within the projects.

By combining these three aspects, we can capture both the quantity (duration and rate) and the quality (type of activity) of a bot's participation. Finally, we also take a closer look at selected individual bots (e.g., bots that commit code), reading through their documentation. This will help us explain and comprehend some effects the bots may have on their projects.

3.2.3 RQ2 – Activity Roles in Developer Networks

With RQ2, we want to understand how developer networks evolve when a bot is introduced to it. To do so, we start by creating developer networks with sliding windows for six-month activity periods across the entire lifetime of each project. We have decided to create networks of six months, because they give accurate overviews of project dynamics over a large chunk of time [22, 26]. With sliding windows we obtain networks that overlap each-other by three months, this allows us to capture trends in developer dynamics that can be lost if they happened between two networks with no overlap. Sliding windows are also very useful for capturing random events such as bot adoptions, allowing to better understand the effects of those events. The types of developer networks we focus on are issue networks (see section 2.4). We believe that co-change networks would not be as relevant because only few bots regularly commit in the code and they "mostly work with configuration and documentation related files" [16]. To examine the context from when a bot has joined, we examine the six months during which they entered the project. Throughout that period we look at different metrics that help us grasp this context:

- Number of core developers, determined as developers with centrality above the 80% threshold (see Subsection 3.2.1)
- Number of bots
- Domination in centrality: Who currently has the highest centrality, whether it is a single or multiple developers, a group of humans or bots.

Meanwhile, for each bot we try to answer the following questions:

- Is their centrality above the 80% threshold typically attained by core developers?
- Is their centrality above the centrality of all human developers?
- If the bots stay active for longer than a year, does their centrality stay consistent?

To compute the centrality metrics of a bot in a developer network, we use eigenvector centrality and hierarchy centrality (see section 2.4), as both imply different levels of centrality. A central developer would have high scores for both metrics. The centrality consistency of a bot is assessed using the standard deviation of its centrality. We choose to study the consistency in centrality of bots that have been active for at least four networks in a row (over one year[3]), because using four windows ensures we capture meaningful patterns in variation as bots active in fewer networks do not have sufficient data points to reliably assess their trends in centrality.

To analyze what factors most influence bot centrality, we use a linear mixed-effects model, in which centrality data from all bots across all projects are combined. This model helps us identify patterns and determine whether specific project characteristics are linked to higher or lower centrality for the bots. We focus on key project metrics as explanatory variables[4]: the number of human developers, the number of core developers, the number of bots, the number of issue events triggered, and the activity type of the bot. These metrics represent the primary factors we examine to understand their impact on centrality. This is the model formula:

$$\begin{aligned}\text{centrality} \sim\ & \text{developer\_count} + \text{core\_developer\_count} + \text{bot\_count} + \text{events\_count} \\ & + (1|\text{name/project} + \text{activity\_type})\end{aligned}$$

The terms between brackets are random effects, individual terms that are unique for each bot and do not change for them from one network to another, while the other effects are fixed effect coefficients which change for them from one network to another. Because some bots might be present in several projects, we identify the bots by both their project and their name, hence the notation "name/project" in the formula above. The core developer count was assessed by counting the numbers of developers that had reached the 80% threshold in eigenvector centrality. We build two separate models—one for hierarchy centrality and another for eigenvector centrality—to explore how these project features shape bot centrality.

3.2.4 RQ3 – Project Metrics Impact

To assess whether a bot has influenced a project, we compare different metrics from periods before and after its adoption. Based on the literature [10, 40, 43, 45], a time-frame of about 12 months before and after bot adoption seems appropriate to compare how different project metrics change post-adoption. Figure 3.1 illustrates which time frames we consider based on certain events happening around bot adoption. (A) shows the regular before and after time-frame when no bot adoption occured around the time a bot has been adopted. To account for time frames during which multiple bots have been adopted, we group these bots together and consider the 12 months before the first bot adoption of that group and the 12 months after the last bot adoption from that group as illustrated Figure 3.1 (B). However, if two bot adoptions are separated by less than a year but their individual 12-month time

3 One year of activity corresponds to three overlapping six-month networks, with each new window starting three months after the previous one (e.g., Jan–Jun, Apr–Sep, Jul–Dec).

4 These metrics were selected as they are the most likely to correlate with centrality. For example, projects with larger numbers of developers may lead to bots triggering more events, thereby increasing their centrality in the network.

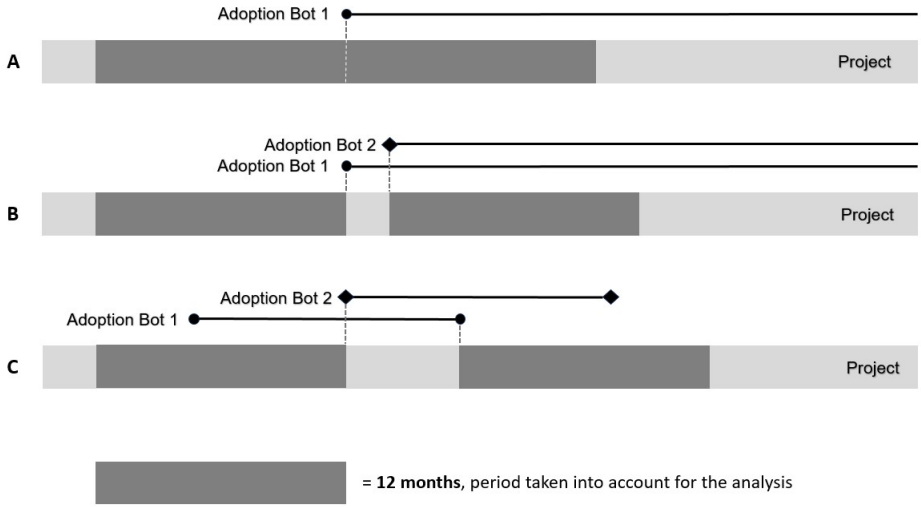

Figure 3.1: Time frames considered around bot adoption
A – Regular bot adoption. B – Grouped bot adoption. C – Bot adoption around bot departure.

frames overlap by less than 4 months, we analyze each bot separately using its own 12-month before-and-after time frame. They do not get grouped up together. If a bot has ceased activity around the same time as when another bot has started its activity, we consider that stopping activity may have had an impact on project metrics. Therefore, we treat the bot stopping activity like a bot adoption and group it with chronologically adjacent bot adoptions. Figure 3.1 (C) illustrates this case. Because of the nature of that role, we ignore bots labeled as "Trial" in RQ1 (subsection 3.2.2).

To compare our results with previous work and understand differences that activity roles bring, we decided to look at pull-request metrics like Wessel et al. [40, 42]. In addition, we want to study other metrics such as the issue resolution rates for non-PR issues, the number for each event type triggered by bots and those triggered by human developers. Similarly to the literature, we are conducting time-series analysis to allow comparison of the outcome variables trends before and after bot adoption. In contrast to previous work, we are not using RDD as a means to observe the changes after bot adoption, as it would require significantly larger datasets than those available for this study. We end up considering the following different metrics collected for each month:

- Number of PRs that have been merged.
- Number of PRs that have been rejected.
- The median time (in hours) it took to close PRs (starting from when it was opened until when it was merged/rejected).
- Median number of comments made by human developers.
- Median number of human comments per issue.
- Number of resolutions for non-PR issues.

- Median time (in hours) for non-PR issue resolution.

To assess whether there were significant changes in the different metrics due to bot adoptions, we fit a linear model for each metric. This model helps us identify the source of these changes, since they could stem from factors such as an increasing number of developers, pre-existing trends, or the bot adoption itself. Identifying these sources is important, as changes cannot be automatically attributed to bot adoption. For each bot adoption group, we therefore fit a linear model for each metric, using the following formula:

$$\text{metric} \sim \text{index} + \text{after\_adoption} + \text{index:after\_adoption} + \text{number\_of\_developers}$$

Our models follow a segmented regression specification commonly used in interrupted time series designs [3]. Here is what each term represents:

- `index`: The distance (in months) from the bot adoption, with negative values indicating months before adoption and positive values indicating months after.
- `after_adoption`: A binary variable (false before adoption, true after adoption) capturing the immediate effect of bot adoption. We sometimes refer to changes measured by this term as "level" changes.
- `index:after_adoption`: The interaction term, which accounts for changes in trends after adoption. We sometimes refer to it as "trend" or "slope".
- `number_of_developers`: With this term we control for the influence of developer activity on the metric. It represents the monthly count of unique developers who triggered events in the repository.

This model helps us isolate the effects of bot adoption by taking into account pre-existing trends (index), post-adoption trends (index:after_adoption), immediate adoption effects (after_adoption), and developer activity (number_of_developers). For each term, we report the partial R^2, which quantifies how much of the observed change is attributable to that specific factor. As this model is known to induce collinearity [3] — `index`, `after_adoption` and `index:after_adoption` "are not dependent" [3] — we want to evaluate the model carefully. This is why we use a Mann-Whitney U test, to understand if there are any significant changes between before and after bot adoption and we rely on Cliff's delta to assess effect size.

We have noted that the term `after_adoption` captures immediate changes following bot adoption. However, when multiple bot adoptions occur with significant time gaps (greater than one month), this term may inappropriately attribute project metric changes solely to the adoption event, rather than to the passage of time between adoptions. To address this, we exclude `after_adoption` when analyzing multiple bot adoptions. This adjustment has two purposes: it reduces multicollinearity (because `after_adoption` is linked to `index:after_adoption`) and prevents the model from overestimating the immediate impact of bot adoption by misattributing changes that are actually due to the time elapsed between adoptions.

The exclusion of this term does not compromise our ability to assess the effects of bot adoption on project metrics. Instead, we rely on the `index:after_adoption` interaction term,

which is more informative as it indicates any long-lasting changes following bot adoption. In cases where multiple bots are adopted in closer succession (within the same month), we keep the `after_adoption` term to capture any immediate effects.

4

Evaluation

In this chapter, we present the results of our exploratory study on bot activity. The chapter is organized around our research questions, each addressed in a dedicated section. Within each section, we present the key findings derived from our analysis, followed by a discussion and an interpretation of these results to answer the corresponding research question.

4.1 Determining Bot Activity (RQ1)

In this section, we introduce the bots detected in the projects and address our first research question:

> *"What roles do bots take in GitHub projects, and which specific tasks do they perform?"*

We begin by identifying the bots present in each project and describe bot-project interaction. Next, we analyze the distribution of activity duration and rate among the bots. We then classify the bots into activity types based on the issue events they triggered. Following this, we examine specific bots in detail, using their documentation to explain their roles and assess the extent to which their activities align with these roles. Finally, we discuss these findings and how they contribute to answering our research question.

4.1.1 Projects and Bot-identification

We collected data on ten OSS projects of diverse scales, ranging from tens to thousands of contributors: Bootstrap, Ghost, Gogs, Google Data Transfer Project (Google-dtp), jQuery, Keras, Next.js, React, Redux, and Vue.

By using the bot identification method specified in Section 3.2.1, we were able to label 128 contributors as bots. We found that 35 of those contributors had human-like profiles and interactions. As far as we know they seem to have been mislabeled because of their commits that were following a template similar to those bot sometimes use. An example of this can be seen Figure 4.1.

Some bots were active once or for short amount of times. Because of how numerous they were, we decided to define two categories of those bots seperating those with minimal activity and those with slightly higher, but still limited, activity. *Trial* bots were active for less than a month or had triggered less than 5 events in total, *extended trial* bots were active for less than 2 months or had triggered less than 50 events in total. Throughout the 10

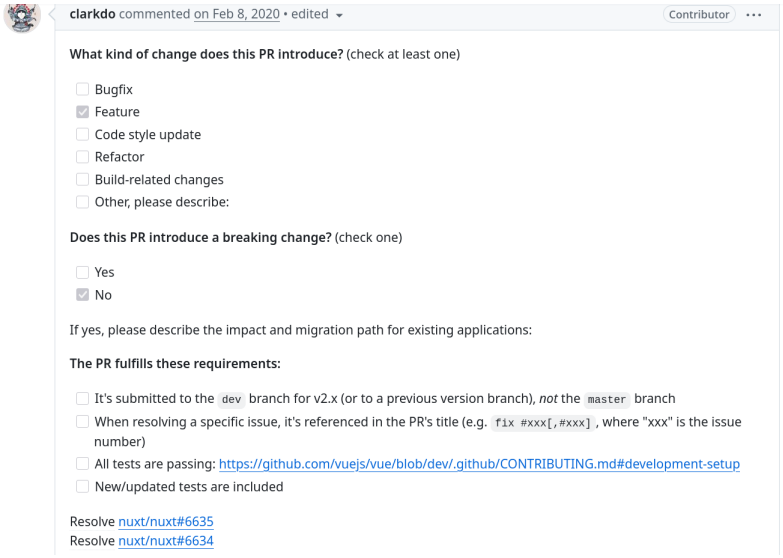

Figure 4.1: Example of human wrongly labeled as bot adding a commit using a template in Vue.

projects, we found 35 trial bots and 15 extended trial bots. Resulting as a total of 50 bots out of 93.

5 extended trial bots and 18 trial bots exclusively triggered the events `mentioned`, `referenced` and `referenced_by`. These bots ultimately did not contribute to any of the projects directly, either being mentioned or referenced by a project contributor, or referencing an issue of that project while contributing to a different project.

To better grasp the activity levels of each project, we considered the number of developers active in each six-month window from our developer networks (see Section 3.2.3). Table 4.1 presents the projects, the mean number of contributors across all 6 months windows, and the number of bots we identified. Because our data was collected at different times, we also have different observation periods for each project.

Table 4.1: Project Overview

Project Name	Bot Count	Mean Contributor Count	Observation Period
Bootstrap	16	≈ 1,739	2011-08-19 – 2021-01-07
Ghost	7	≈ 331	2013-05-07 – 2023-06-19
Gogs	14	≈ 372	2014-02-14 – 2023-03-06
Google-dtp	2	≈ 22	2018-01-04 – 2020-05-01
jQuery	8	≈ 191	2010-09-03 – 2020-12-24
Keras	4	≈ 1,809	2015-03-28 – 2020-03-17
Nextjs	11	≈ 1,815	2016-10-06 – 2021-01-10
React	12	≈ 1,392	2013-05-29 – 2020-12-24
Redux	5	≈ 429	2015-06-01 – 2020-12-23
Vue	7	≈ 950	2013-09-08 – 2020-12-22

In the appendix A.1, we added an overview of the bot activity across each project. Our projects span a great set of OSS repositories. Some, like React, Nextjs, Bootstrap, and Vue, involve thousands of contributors and rely heavily on bots for testing, integration, and ongoing maintenance. Others, such as Ghost, Gogs, and the Google-dtp, are smaller in scale, with fewer contributors and lighter automation. This variety makes it possible to study different patterns in developer dynamics and the ways project metrics develop. For Google-DTP we only obtained a short observation period of just over 2 years. This is the smallest project, with a developer count averaging around 22 and 2 bots with low activity. We also noticed bots with similar names, predominantly *dependabotbot* and *dependabot-previewbot* in Gogs and Bootstrap, and *codecovbot* and *Codecov Comments Bot* in Ghost, everything seems to indicate that these bots changed names but are shown as two different contributors by Github. Some bot aliases also appear with a capital letter, namely *Dependabot* and *Coveralls*. The bots with these names would only trigger `mentioned` events, due to someone linking the dependabot or the coveralls webpage. The only project that had two bots with the same name and different activity is Nextjs with *Vercel* and *vercelbot*.

4.1.2 Activity Duration and Rate

We calculate activity duration and activity rate distribution by excluding the trial and extended-trial bots, using the remaining 43 bots. For duration, the median is at 18 months, Q1 at 8 and Q3 at 29.5. For activity rate, the median is at 65.1 triggered events per months, Q1 at 29.6 and Q3 at 220.1. Table 4.2 shows how we ended up dividing these aspects into groups.

Table 4.2: Activity Duration and Activity Rate Classification

Duration group	Duration	Rate group	Activity rate
Short	> 1 and ≤ 8	Light	≤ 30
Mid-lengthed	> 8 and ≤ 29	Medium	> 30 and ≤ 120
Long	> 29	Intense	> 120

Figure 4.2 shows how bot activity rate is distributed amongst the projects. We have excluded trial and extended-trial bots from this figure. We observe that the activity rate of bots does not appear to be linked with the number of contributors in the projects. Bootstrap and Ghost have similar bot counts (8 and 7 bots respectively), and while Bootstrap has thousands of contributors and Ghost hundreds, the latter project has more bots with intense activity rates.

4.1.3 Bot Activity Type

Observing the different issue events triggered by bots (see Appendix A.1), leads us to find that certain bots would trigger a higher diversity of events than others. This observation suggests a hierarchical structure in bot permissions, where bots capable of triggering higher-

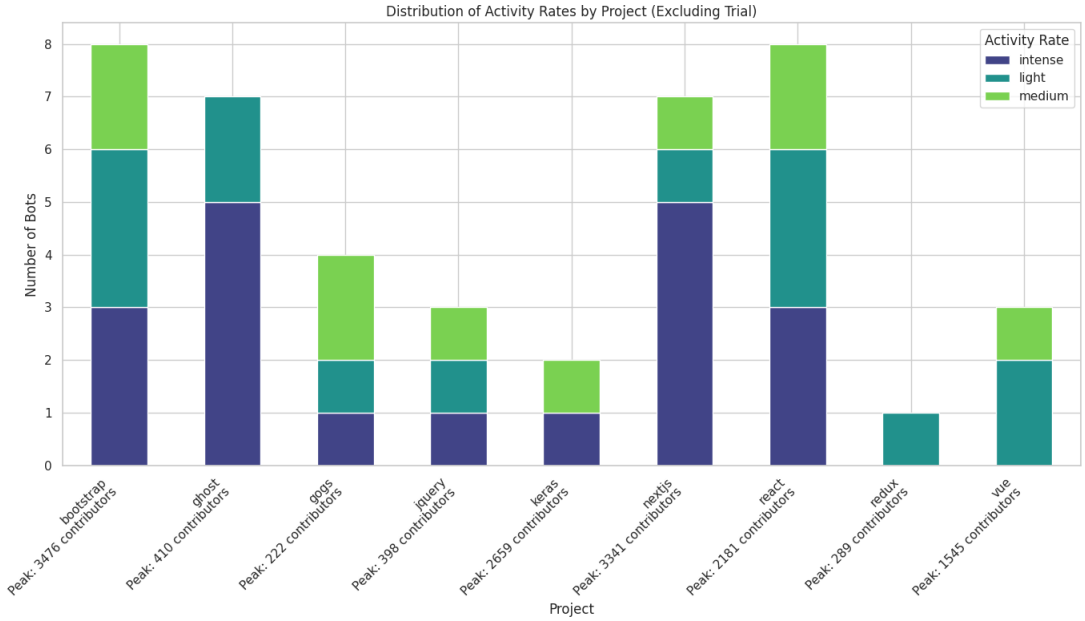

Figure 4.2: Activity rate groups per project.
The peak developer count for six month time-frames during which bots were active is indicated bellow project name.

permission events (e.g., adding commits) also tend to trigger lower-permission events (e.g., commenting or labeling issues). To capture this hierarchy, we selected events that represent distinct ways bots interact with projects, reflecting different permission levels. Table 4.3 presents the five inferred activity types and which events a bot has triggered to be attributed one of the activity types. Bots from activity types at the top of the table (Code Management, State Management) typically trigger events from lower level activity types as well. Though most state management bots like *Lockbot* trigger exclusively `locked` and `unlocked` events.

Table 4.3: Activity Type Attribution Rules for Bots

Activity Type	**Triggered Issue Events (first match rule)**
Code Management	`commit_added`, `head_ref_force_pushed`, `merged`, `head_ref_deleted`
State Management	`locked`, `unlocked`
Label Management	`labeled`, `unlabeled`
Commenting	`commented`, `subscribed`, `add_link`
Referenced or Mentioned	`referenced`, `referenced_by`, `mentioned`

We have found that bots will generally trigger all events that match their activity type. With 38 bots, the commenting activity type is the largest group out of the five. Code Management is the second largest with 27 bots, while referenced or mentioned bots are a group of 17, label and state management are the smallest group with just 7 and 4 bots respectively. Commenting bots are the most numerous, this interaction type allows them to contribute to the project in many ways. For example, *Coveralls* and *Codecov* post comments

on pull requests or commits with test coverage summaries. *Github-actions*, though it is uniquely configured for every project, and *Codesandboxbot* comment CI feedback or test results.

Some of the commenting bots have very unique contributions. For instance, *IssueHunt Bot* puts bounties on bug issues and rewards developers for solving the bugs, *Mention Bot* comments to notify users or maintainers when they are mentioned in issues or PRs.

4.1.4 Bot Roles

Figure 4.3: Distribution of Bot Activity Types — Number of Bots per Role

Figure 4.3 shows how bot activity types are distributed among the other two activity role aspects. The activity type "Referenced or Mentioned" has bots that had little to no activity, almost exclusively trial bots triggering one or two events. *Borsbot* in React is the only bot that has not been labeled as trial, being part of the short duration and medium activity rate groups. *Borsbot* is now documented as a Github App that is no longer running. What it used to do is batch pull requests to test them together, with the goal of reducing the number of builds and tests required. It was only used by single users working outside this repository on Mozilla extensions, where that bot ended up triggering 106 `referenced` events by referencing React PRs in just 4 commits. We see here, that though label and state management are the smallest groups, they often have intense activity rates. Tables detailing the three role aspects of each bot for every project are included in Appendix A.2. We use the information from these tables in this chapter when discussing the characteristics of specific bots.

4.1.5 Discussion

When examining activity rate, we expected there would be a strong correlation between the number of active contributors in the project and the activity rate of its bots. However, what we have found suggests that bot activity rate is rarely linked to indicators such as developer count. Only projects with very small numbers of developers (e.g., Google-dtp which had at most 41 contributors over 6 months) show bots that fail to reach even light activity rates, and this might only be due to our short observation period and the kinds of bots they have adopted. Projects like Vue reached over a thousand active contributors in some time-windows, but no bots have achieved high activity rate, whereas Ghost had just over four hundred developers at its peak, and hosted bots with very high activity rates with *codecovbot* or *Renovate Bot* accounting for over a fourth of the total triggered events in certain windows. In Gogs, *github-actions* would trigger more than half of the total events at certain periods of time. This shows that bot activity cannot be simply explained by the overall project activity, or by the types of events a bot triggers. Instead, each bot seems to follow their own activity profile, sometimes unique to the project they are contributing in. *Codecovbot* and *github-actions* both are present in other projects in which they exhibit significantly less activity. Referenced or mentioned bots appear in the timeline, but their involvement is tied to an external activity because they do not contribute to the project directly, whereas dependency management bots (see Section 2.1), such as *Dependabot* or *Greenkeeper*, have a steady but moderate activity that is more influenced by the rhythm of dependency updates than by developer activity. Figure 4.3 also reads that, except for "Referenced or Mentioned", activity types are very much independent of activity rate and duration.

Overall, it seems that how intense or light bot activity is, does not just depend on how active a project is, it also depends on what work the bot is supposed to perform, how it wants to perform it, and how the project has configured it.

Answer to RQ1

Bots in GitHub projects take on diverse roles that can be grouped into five main activity types: code management, state management, label management, commenting, and external referencing. Their activities range from high-permission tasks such as committing and merging code, to supportive roles like posting test results, labeling, or locking issues, to peripheral activity tied to external references. Most bots fall into the commenting category, while code management bots are fewer but more impactful due to their higher-level permissions. Importantly, our analysis shows that the intensity and duration of bot activity cannot be explained solely by project size or developer count; rather, it depends on the specific function of the bot and how it is configured within each project. Thus, bots do not just automate repetitive tasks, but contribute in different ways that reflect both their usage purpose and project-specific configurations.

4.2 Bot Roles and Developer Networks (RQ2)

For each project, we created developer networks with sliding windows spanning over our entire observation periods. The goal is to understand bot activity and developer network interaction, relying on the bot roles defined in Section 4.1. In this section, we try to answer our second research question:

"How are these bot roles reflected in the structure of the project's developer network?"

We begin by assessing how central bots are compared to human contributors. We then assess who are the most central actors in the networks, seeking to understand if and how bots may have the highest centrality of all contributors. Afterwards, we evaluate what may make bot centrality fluctuate. Finally, we discuss these results.

4.2.1 Bot and Human Centrality Comparison

We observe that bots seem to almost systematically reach the 80% centrality threshold typically attained by core developers. Even trial and extended-trial bots reached that threshold, indicating they were among the 20 most central contributors, sometimes after triggering just a single event. However, all trial bots that only added commits, and some trial commenting or referencing bots, did not reach it; all other bots surpassed the threshold if they triggered `commented`, `mentioned`, `referenced`, or `referenced_by` events.

Lockbot in NextJS triggered thousands of `locked` events across each of the networks but only reached the threshold on the rare occasions when it commented. *Vercel*, in that same project, triggered `locked` and `comment_deleted` events with extremely light activity (48 comment deletions and 59 issue lockings in over three years) and nevertheless crossed the threshold every time. While comment deletions seem to affect centrality positively, locking issues appears to have no effect.

The `mentioned` event, by contrast, can lead to very high centrality. We found that in the React repository, *CodeSandbox* obtained a hierarchy centrality score of 6 and *dependabotbot* a score of 5. Both crossed the threshold, but while *dependabotbot* triggered "created," "commented," and "labeled" events 19 times each, *CodeSandbox* reached a higher centrality score with only a single "mentioned" event. *CodeSandbox* did not participate in the project in any other way, nor did it contribute at all. What happened here is that another bot, an alias to *CodeSandbox* named *codesandboxbot*, added a link to their website in an issue that involved many highly central contributors. This automatically mentioned *CodeSandbox* and created network links with project maintainers.

A final observation concerns Google-dtp, the project with the smallest contributor count. Unlike in most other projects, where bots that contribute through comments almost systematically cross the centrality threshold, neither of the two bots in this repository did so consistently. Only *googlebot* occasionally reached the threshold. Overall, both bots in that project showed very light activity.

4.2.2 The Most Central Actors

There are no projects in which bots fail to reach high positions in network centrality, as they always rank among the top 10 developers. In fact, they share the very top positions—consistently ranking first, second, or third in 6 out of 10 projects. We will refer to being among the 10 most central contributors of a network as reaching or achieving high centrality.

In Keras, Gogs, and Ghost, some bots even hold the most central positions, remaining more central than any human developer. In Keras, the label-management bot *Stalebot* had only a short period of activity, yet completely dominated the networks while it was active. Its main activity consisted of: `commented = 2154`, `labeled = 3278`, and `state_updated = 2839`. In Gogs, two bots occupied the top positions in hierarchy centrality. A human contributor did manage to rank higher than both bots several times, holding the top spot in eigenvector centrality, but the bots maintained second place in that metric. One of these was *Codecovbot*, an intense-activity commenting bot that also triggered a large number of `referenced` events. The second was *Dependabot*, which triggered a wide variety of events, though at a much lower activity rate than *Codecovbot*. The presence of *Dependabot* among the top-ranking bots in this project demonstrates that a code-management bot does not necessarily need an intense activity rate to achieve extremely high centrality. Interestingly, *github-actions* did not obtain particularly high centrality scores although, in certain time windows, it triggered more than half of the total triggered events, mostly `locked` events. In Ghost, bot dominance of centrality is evident in eigenvector centrality. Table 4.4 shows an overview of which bots in Ghost attained high centrality, in how many networks and their best and median rank. *Codecovbot*, *Renovate Bot*, and *Ghost CI* are among the most central contributors. They share the three highest positions in eigenvector centrality. All three exhibit intense activity rates, and all three post a large volume of `commented` events, with *Renovate Bot* also creating a lot of commits (almost 20,000 `head_ref_force_pushed` events in the span of four years). *Renovate Bot* did not immediately reach the top ranks after its introduction but, after a little more than a year, began to dominate the networks, sharing the position of most central contributor alongside *Codecovbot* (introduced later). Interestingly, *Renovate Bot* does not appear in the top 10 when considering hierarchy centrality. Instead, *Ghost CI* is far more central in that centrality type, with *Codecovbot* slightly less so. With some human contributors hierarchy centrality values comparable to these bots, the position of most central contributor in hierarchy centrality is contested by humans.

In three other projects, React, Redux, and Vue, bots achieved high centrality but never fully displaced human contributors from the top-most positions. In React, the *Facebook Community Bot*, a label-management bot, consistently held the second-highest centrality across most networks. Its activity was characterized mainly by large numbers of `commented` events, and non-negligible `state_updated` and `referenced` events. Despite this strong presence, human contributors always occupied the top spot. In Redux, the *Netlifybot*, a commenting bot, reached very high centrality values, regularly appearing among the top three contributors, though it never surpasses the leading human developers. Interestingly, *github-actions* appeared in the top 10 of eigenvector centrality across three networks, despite triggering only seven `commented` events during its seven months of activity. We saw that its interactions were with *Netlifybot* and some of the most central human contributors. Finally, in Vue, the *Vue-bot*, a code-management bot, regularly ranked as the second-most central contributor

Table 4.4: Overview of bot centrality rankings in the Ghost project.

	Hierarchy				Eigenvector			
Bot	**Net.**	**Median Rank**	**Best Rank**	**Mean Cent.**	**Net.**	**Median Rank**	**Best Rank**	**Mean Cent.**
stalebot	12	7	5	180.2	5	8	7	0.15
label-actionsbot	10	5	3	512.1	3	9	9	0.12
Ghost CI	7	1	1	5485.5	6	3	3	0.64
codecovbot	7	4	1	379.0	4	1	1	0.97
Codecov Comments Bot	–	–	–	–	3	4	4	0.33
Renovate Bot	–	–	–	–	9	1	1	0.79
greenkeeperio-bot	–	–	–	–	5	6	5	0.28

Net. indicates in how many networks the bot was present in the top 10.
Dashes – indicate the bot has not been present in the top 10 in any network.
Lower ranks are more central.

across most networks for both hierarchy and eigenvector centrality. Its activity consisted of high amounts of `commented` events, placing it in an influential position, yet still behind the most central human contributors.

In the projects where humans remained at the top of hierarchy, we found that the bots that reached high centrality were largely commenting bots. In Bootstrap, bots never attained the very top positions, but appeared in the top ten almost consistently, in networks otherwise clearly dominated by a few human contributors. The bots that reached these ranks were mostly commenting bots, with the exception of *Dependabot*, an intense code-management bot that triggered a wide variety of events (mostly `labeled`, `review_requested`, `commented`, and `created`). Another case is the *Travis bot*, which was active for only three months but managed to rank higher than any other bot, largely due to its activity consisting of `add_link` and `commented`. In Google-DTP, there were constantly less than 10 developers above the 80% threshold due to the small contributor count. Bots never came close to the most central positions, with *Googlebot* occasionally reaching the lower edge of the top ten but never challenging the most central human contributors. In jQuery, two bots briefly reached the very top positions, but each only once, leaving overall centrality dominated by humans. *Lockbot* showed the most intense activity rate among bots in this project, but with activity restricted to locking and unlocking issues it did not keep consistent high centrality. Bots that reached higher ranks in jQuery typically managed to do so through a mix of `commented` and `mentioned` events. Finally, in Nextjs, *Kodiakbot* reached high centrality despite producing only a small number of `commented` events, but it triggered a lot of `referenced`, `merged`, and `state_updated` events. *Github-actions* also showed very intense commenting activity for seven months, reaching second place in centrality during that time.

Taken together, three patterns of bot centrality metrics can be observed in these projects. In some projects bots frequently dominated the networks, reaching the highest centrality values. In others, bots achieved consistently high centrality but remained in second place, never fully topping the most central human contributors. In the remaining projects human

dominance was clear, with bots only sporadically reaching high ranks. Across all three patterns, we observe that bots achieve high centrality by primarily triggering events related to communication: `commented`, `mentioned`, `referenced`. Having an intense activity rate increases chances for a bot to obtain the most central positions, but interactions with larger and more central groups of contributors also increases these chances. Having an intense activity rate increases the likelihood that a bot will obtain the most central positions, but interactions with larger and more central groups of contributors also how central a bot is. We also note that, although referenced or mentioned bots achieved surprisingly high centrality relative to their light activity rates (often reaching the 80% threshold), they remained absent from the top ten positions in any of the networks.

4.2.3 Consistency in Bot Centrality

We evaluated whether bot centrality is consistent across time by computing the variance of their eigenvector and hierarchy centrality. We found that overall, the centrality of bots stay very consistent over their activity time. But hierarchy centrality, even after adjusting its scale, was a lot less stable than eigenvector centrality. In fact, the mean variance for bot hierarchy centrality we measured is of 0.0235, which is 17 times higher than the mean variance of 0.3972 for hierarchy centrality. This variance is very present with bots that have short activity duration and intense activity rates, while in general extended trial and code management bots also suffered from moderately less consistent centrality.

Table 4.5: Linear mixed-effects models predicting bot centrality.

	Eigenvector Centrality			Hierarchy Centrality		
Predictor	Estimate	Std. Error	p-value	Estimate	Std. Error	p-value
Fixed effects						
Intercept	0.170	0.051	0.007**	0.332	0.679	0.652
Human developers	-0.024	0.016	0.117	0.089	0.068	0.191
Core developers	-0.0001	0.0001	0.319	0.0015	0.0006	0.015*
Number of bots	0.0019	0.005	0.718	-0.006	0.022	0.776
Total activity	0.016	0.012	0.193	-0.005	0.053	0.929
Random effects variance						
Project	0.011			0.477		
Bot name	0.031			0.000		
Activity type	0.003			2.114		
Residual	0.025			0.395		

* indicates $p < 0.05$ (statistically significant)
** indicates $p < 0.01$ (highly statistically significant)

We applied linear mixed-effects models using both centrality types as dependent variables. The predictors included the number of human developers, the number of core developers, the number of bots, and the bot's activity type, with project identity specified as a random effect. Results are presented in Table 4.5. For eigenvector centrality, the model showed no significant effects of any predictors. Neither the number of human developers, core developers, nor bots, nor the activity type of the bot, explained meaningful variance. Activity type in particular accounted for only a negligible share of the variance (0.0029). The findings for hierarchy centrality were different. Here activity type explains a significant part of the variance (2.10). Bot identity Additionally, we found a statistically significant effect of the number of core developers ($p = 0.015$) with a very small effect size (estimate = 0.0015).

Overall, we notice a consistent centrality for all bots that is not much affected by any project metrics. We notice differences in how hierarchy and eigenvector centrality are affected, eigenvector centrality emerging as a more stable measure. And we relate centrality variance to some activity roles, mainly roles with shorter more intense activity and code management bots.

4.2.4 Discussion

Our results show the different ways bot roles affect their positions in developer networks. First, it is clear that not all events contribute equally to centrality. We observed that events such as `commented`, `mentioned`, `referenced`, `referenced_by`, and `add_link` are strongly associated with higher centrality because they directly connect bots with other contributors. By contrast, `locked`, `unlocked`, `labeled`, and `unlabeled` events had little to no effect, this suggests that bots restricted to state- or label-management activities rarely become central actors. This difference highlights that centrality is not only a matter of how many events a bot triggers, but of whether these events create meaningful ties in the project's social hierarchy.

This is illustrated by several cases. In Redux, *Github Actions* reached the top ten in eigenvector centrality despite producing only a handful of `commented` events. Its high rank can be explained by the fact that these few interactions connected it with both *Netlifybot* and the most central human contributors, disproportionately increasing its centrality. Similarly, in React, the *CodeSandbox* bot reached high hierarchy centrality from a single mention in an issue that involved some highly central maintainers. These examples show that bots can achieve high positions not only through intense activity rate, but also strategic placement in the network: connections to already central contributors carry more weight than many peripheral interactions.

High activity rate remains an important factor, however. Bots such as *Codecovbot*, *Renovate Bot*, and *Ghost CI* dominated centrality rankings in Ghost due to their intense activity, which consistently placed them among the most central actors. But even here, we note that high activity rate alone is insufficient: *github-actions* in Gogs, despite triggering more than half of all events during certain time windows, never achieved comparable centrality, probably because its activity consisted mainly of locking events that did not create ties with influential developers.

Project context also has importance in bot centrality. In small projects such as Google-DTP, bots failed to establish sustained central positions. *Dependabot*, for instance, remained peripheral and was even classified as an extended-trial bot, largely because the project contained few dependencies. Here, it may be explained by the networks being disconnected and never reaching a small-world structure. An odd case, considering it is a very small project. In contrast, larger projects such as Keras and Ghost saw certain bots dominate the networks completely. *Stalebot* in Keras, with its large commenting, labeling, and state-updating activity, overshadowed all human contributors while active. However, its noisy presence may explain why it was discontinued and only had a short activity duration.

Our variance analysis suggests that centrality is relatively stable over time for most bots, especially when measured by eigenvector centrality. Bots with shorter but intense activity bursts, extended-trial and code-management bots, exhibited higher variance in hierarchy centrality, but eigenvector values stayed more consistent overall. This aligns with the linear mixed-effects models, where eigenvector centrality showed little variance explained by project- or bot-level predictors, whereas hierarchy centrality was more sensitive to bot activity type and the number of core developers. The small positive effect of core developer count may be explained by the fact that a larger core makes it more likely for bots to interact with highly ranked contributors, thus slightly increasing their own hierarchical positions. It also further reinforces the link between hierarchy and developer centrality, considering the core developer count has been computed based on the number of developers reaching the 80% threshold in eigenvector centrality.

Finally, our results indicate that the way bots are introduced into projects may be decisive. Bots are often adopted and configured by maintainers, which makes interactions with them more likely. We also found that bots frequently interact with one another, this further strengthens their position in central clusters of the network.

Taken together, these findings suggest that bot centrality cannot be reduced to a single factor such as activity type or raw activity levels. Instead, it comes from a combination of three interdependent elements. First, activity rate plays a role: bots that trigger thousands of events, such as *Renovate Bot* or *Stalebot*, are more likely to appear among the most central contributors simply because their volume of interactions creates many ties across the network. However, high activity alone is not sufficient. Second, the type of events matters greatly. Communication-oriented events—such as `commented`, `mentioned`, or `referenced`—tend to create stronger and more central ties than management events like `labeled` or `locked`, which have little impact on network positioning. This explains why some lightly active bots (e.g., *CodeSandbox*, with only one "mentioned" event) can nevertheless achieve high centrality. Finally, network placement is decisive. Bots that interact with already central contributors or other influential bots inherit a central position themselves, even with low amounts of interactions. This is illustrated by the case of *Github-actions* in Redux, which reached high ranks despite a very low activity count because its few interactions linked it to *Netlifybot* and central maintainers.

An important implication of these findings is that networks containing bots should be interpreted with caution. Because some bots reach extreme levels of centrality, they can distort the network structure by overshadowing human contributions and inflating the apparent influence of automation. In these cases, bots may obscure who the true human leaders and coordinators are. Also, humans who interact frequently with bots (for example,

through dependency updates or automated test feedback) may appear more central than they would otherwise be because they interact with bots, who although very central are not leading the project like important maintainers would. An important leader in a project could appear less central if they mostly interact with humans rather than bots. For example, a maintainer making key design decisions might rank lower than a developer who frequently replies to automated dependency updates or CI comments, even though the maintainer plays a far more influential role. This could justify filtering out bots in analyses of developer dynamics, or at the very least analyzing them separately to avoid overestimating their role and underestimating the contributions of certain human collaborators in developer networks.

Answer to RQ2

Our findings show that bots consistently reach high positions in developer networks, sometimes even surpassing all human contributors in centrality. Whether a bot becomes central depends on a combination of three factors: its activity rate, the types of events it triggers (with communicative events such as `commented`, `mentioned`, and `referenced` being especially impactful), and most importantly, whether these interactions link the bot to already influential contributors such as maintainers or other central bots. Short-lived or sporadic bots tend to show more unstable centrality, whereas consistently active commenting and code-management bots often dominate networks.
However, these results also underline a risk of network distortion: When bots dominate networks, they may overshadow human contributions, obscure leadership structures, and artificially inflate the centrality of humans who interact with them. For this reason, networks that include bots must be analyzed carefully. In some cases, filtering bots out or treating them separately may provide a clearer picture of actual human collaboration and governance.

4.3 Effects of Bot Adoption on Project Metrics (RQ3)

In this section, we present the results of our comparison of project metrics before and after bot adoption. We then discuss these findings, explaining and comparing the effects of different bots, and weighing the model's problems. And we answer our third and last research question:

> *"What impact do specific activity roles have on the project after bots have joined them? How do different bot activity roles influence project metrics after bot adoption?"*

To evaluate the impact of bot adoption on project metrics, we analyzed each project's before–after timelines, accounting for developer activity and overall project trends. Table 4.6 presents all the bots for which the before and after trends were analyzed, mentioning which bots have been grouped together. We noted that no bot of the "referenced or mentioned" activity type was active for long enough time to qualify for this analysis, the longest duration of activity being of two months for this type (*borsbot* in React). The dates correspond to

adoption of bots. Multiple dates in the same row indicate multiple adoptions or abandonments that have been grouped together due to their temporal proximity. Each adoption date marks the end of the "before adoption" period and the start of the "after adoption" period used in our analysis. For grouped adoptions, only the first and last adoption dates are shown.

An overview of the significant effects from our linear-models on each specific metric is presented in Table 4.7. Tables summarizing all significant results from our linear models (lm) and Mann-Whitney U tests (MW) with Cliff's delta (Δ) are provided in the appendix (Section A.3 and Section A.4, respectively). These tables present, for each project and bot, the metrics where significant effects were detected, along with effect sizes and significance levels. Results with p-values greater than 0.1 were omitted for easier interpretation and reading experience. For the Mann-Whitney U results, only medium and large effect sizes are shown; results with small or negligible effect sizes were excluded.

4.3.1 Results of the Bot Adoption Time-frame Analysis

Right off the bat, we found that project metrics were largely influenced by the number of developers rather than by bots. A few significant changes identified by the MW test were not accounted for by any terms in the lm. However, individual bots, particularly those with high activity rates, still frequently exhibited unique effects.

Code management bots were most likely to have significant changes in pull request metrics. In jQuery, *jquerybot* adoption had an immediate effect on a large increase in PR merges (lm: estimate = 2.49, $p < 0.05$, R^2 = 0.26). In Keras, *TensorflowButlerbot* is linked to decreases in PR merges and rejections (MW: $p < 0.05$, Δ = -0.97 and -0.99, large effect), as well as in issue resolutions (MW: $p < 0.05$, Δ = -1.00). In Ghost, *Renovate Bot* produced large increases in merges (lm: estimate = 10.58, $p = 0.054$) and a strong decrease in PR-closing time (lm: trend estimate = -3.17, $p < 0.05$, R^2 = 0.49). In several cases, like the adoption of *greenkeeper* in Ghost, code management bots had little significant effect.

Label management bots showed consistent effects, particularly in issue resolution. In Keras, *stalebot* increased issue resolution counts (lm: estimate = 17.62, $p = 0.087$) and lengthened resolution times immediately after adoption (lm: estimate = 3211.56, $p < 0.05$, R^2 = 0.46). The increase in resolution time was extremely large and consistent with the bot's purpose, as it closed many old issues abandoned by contributors. In Ghost, the combination of *stalebot* and *label-actionsbot* had large effects on merges, PR resolution times, and comment activity (MW: $p < 0.05$, Δ large for several metrics). In React, by contrast, *stalebot* had only negligible impact and the immediate effect of its adoption was not measured as it was adopted around the same time as two other bots.

We observed diverse effects with commenting bots. In redux, *netlifybot* contributed to a decrease in non-PR issue resolutions (lm: trend estimate = -0.97, $p = 0.082$). In Vue, *Codecov* was associated with longer PR-closing times (lm: trend estimate = 0.20, $p < 0.05$, R^2 = 0.20), consistent with the closure of older PRs. In Ghost, *codecovbot* and *ghost CI* together produced a large increase in comments (lm: trend estimate = 0.41, $p < 0.05$, R^2 = 0.41). Commenting bots with light activity rate, such as *JSF CLA Assistant* in jQuery or *BootstrapIssueCloser* in

Table 4.6: Summary of bots studied in each project. Date(s) column explained in Section 4.3.

Project	Bots Studied	Date(s)
Redux	Netlifybot	2018-10-08
JQuery	JqueryBot	2014-12-03
	Mention Bot	2015-12-03
	JSF CLA Assistant	2016-11-07
	LockBot	2018-06-17
Keras	StaleBot	2017-05-23
	Tensorflow-Butlerbot	2019-10-18
Vue	Codecov	2016-08-11
	vue-bot	2017-03-18
	dependabot	2019-10-13
Google DTP	dependabotbot	2019-05-29
Gogs	Codecovbot / Dependabot (grouped)	2020-02-29 / 2020-03-27
	GithubActions	2022-03-15
Nextjs	Vercel	2017-08-07
	Kodiakhbot (ignoring github actions)	2019-03-05 / 2019-07-25
	Lockbot / End of Greenkeeper (grouped)	2018-05-10 / 2018-05-23
	vercelbot / End of lockbot (grouped)	2020-06-12 / 2020-06-24 until 2021-01-10[1]
Bootstrap	First three Bootstrap bots (grouped)	2014-06-16 / 2014-11-20
	Issue Closer / Hound (grouped)	2015-07-28 / 2015-12-08
	Dependabot	2019-10-16
React	Facebook Community Bot	2014-04-25
	React Community Bot	2015-04-17
	Coveralls	2016-04-28
	Pull Request Analyser / End of ReactCB (grouped)	2018-01-18 / 2017-10-06
	Dependabot / Stalebot / codesandboxbot (grouped)	2019-08-13 / 2020-01-09[2]
Ghost	Greenkeeperio	2016-02-08
	Stalebot, Label-actionsbot, Renovate Bot (grouped)	2019-01-24 / 2019-04-08
	Ghost CI / Codecovbot / End of label-actionsbot and stalebot (grouped)	2021-09-10 / 2021-12-06

1 The project data from our observation period ended on 2021-01-10, before the end of the twelve month period.
2 For grouped adoptions with three bots, we picked only the dates of the first and last adoption chronologically.

Table 4.7: Overview of metrics with significant positive or negative effects, grouped by project and bot.

Metric	Effect	Projects/Bots
Merges	positive	JQUERY: JqueryBot; KERAS: Tensorflow-Butlerbot; VUE: dependabot; GOGS: Codecov / Dependabot; NEXTJS: Lockbot / End of Greenkeeper; BOOTSTRAP: Dependabot; REACT: Coveralls
Merges	negative	GOGS: Codecov / Dependabot; NEXTJS: End of vercelbot; REACT: Facebook Community Bot; REACT: React Community Bot; REACT: Dependabot / Stalebot / codesandboxbot; GHOST: Stalebot / Label-actionsbot / Renovate Bot
Rejects	positive	VUE: dependabot; NEXTJS: Kodiakhbot; REACT: React Community Bot
Rejects	negative	KERAS: Tensorflow-Butlerbot; VUE: Codecov; GOGS: GithubActions
Cmt_tot	positive	KERAS: Tensorflow-Butlerbot; GOOGLE-DTP: dependabot-bot; NEXTJS: Kodiakhbot; NEXTJS: Lockbot / End of Greenkeeper; GHOST: Ghost CI / Codecovbot
Cmt_tot	negative	JQUERY: JqueryBot; GOGS: Codecov / Dependabot; NEXTJS: End of vercelbot; BOOTSTRAP: BootStrap Bots; GHOST: Stalebot / Label-actionsbot / Renovate Bot
Cmt/iss	positive	BOOTSTRAP: Dependabot; REACT: React Community Bot
Cmt/iss	negative	BOOTSTRAP: BootStrap Bots; REACT: Coveralls
PR_time	positive	VUE: Codecov; VUE: dependabot; GOGS: GithubActions; NEXTJS: Kodiakhbot; NEXTJS: End of vercelbot; REACT: React Community Bot; GHOST: Stalebot / Label-actionsbot / Renovate Bot
PR_time	negative	REACT: Facebook Community Bot
Res_time	positive	JQUERY: JSF CLA Assistant; KERAS: StaleBot; VUE: Codecov; VUE: vue-bot; VUE: dependabot; REACT: Pull Request Analyser / End of ReactCB
Res_time	negative	JQUERY: MentionBot; KERAS: StaleBot; GOOGLE-DATA-TRANSFER-PROJECT: dependabotbot; GHOST: Stalebot / Label-actionsbot / Renovate Bot

Metric short names: Merges = merged PR count; Rejects = rejected PR count; PR_time = median PR closing time; Cmt/iss = median comments per issues; Cmt_tot = comments count; Res_count = non-PR issue resolutions count; Res_time = median issue resolution time.

BOOTSTRAP, had no discernible impact. Overall, commenting bots appear more likely to influence discussion metrics rather than closing or resolution metrics.

State management bots had comparatively smaller effects. *Lockbot* in JQUERY and NEXTJS showed little to no measurable influence on merge or resolution metrics. In REACT, *Facebook Community Bot* and *React Community Bot* produced only slight changes in merge and rejection counts (MW: $p < 0.1$, Δ medium), without substantial shifts in other metrics. This suggests

that while state management bots enforce repository maintenance policies, their adoption rarely translates into large changes in the project-level indicators considered here.

Overall, the results indicate that the effects of bot adoption on project metrics are very context-dependent and are often explained by project trends unrelated to bot adoption. Code management and label management bots showed the strongest and most consistent impacts, particularly on pull-request processing and issue resolution. Commenting bots and state management bots, by contrast, tended to produce more variable effects. The intensity of activity rates often related to stronger and more significant changes.

4.3.2 Discussion

The number of developers often explains more variance in merges, rejects, comments, and non-PR resolution than the bot-related terms. Across all projects and metrics, developer activity (number of developers) explained a substantial share of the variance in outcomes. Specifically, in our regression models, the partial R^2 for the developer count term was greater than that of the bot adoption terms in over 80% of cases where significant effects were observed. In contrast, significant changes attributable to bot adoption (either immediate or trend effects) were identified in about 15–20% of analyzed cases, and these were concentrated among code management and label management bots with high activity rates. This means that most significant changes around bot adoptions are due to increases or decreases of developer participation. When Mann-Whitney U indicates significant changes, but the linear regression assigned small R^2 to the bot terms, it is most likely that bot adoption coincides with less developer activity than the bot adoption being the sole cause.

The model failed to show the impact of certain group adoptions such as the *dependabotbot*, *codesandboxbot*, and *stalebot* adoption in React. Two of the three bots had intense commenting activity, but the model was not able to reflect it, likely because the after_adoption term was removed due to the large time between the bot adoptions. No variation in trend was captured because the bots kept a constant activity. The model in general would rarely capture increasing comment counts for commenting bots, except the group adoption with *codecovbot* in Ghost that indicated a great increase in that metric (lm: trend estimate = 0.41, $p < 0.05$, $R^2 = 0.41$).

The model seems to fail in other situations: *dependabotbot* in Vue (a code management bot) has had a positive and significant effect in merge count (lm: estimate = 10.58, $p = 0.054$ in level, and estimate = 3.17, $p < 0.05$ in slope), but MW and Δ indicate a large and significant decrease in merge count (MW: $p < 0.05$, $\Delta = -0.61$). Oddly, no other term in the model explains a decrease in that metric. How this can be interpreted is a likely multicollinearity in the model, flipping the estimate signs for the bot adoption terms. Interestingly, the exact opposite happens in *Gogs*, where another *dependabot* has negative estimates for merge count (lm: estimate = -4.23, $p = 0.061$), but a positive Δ (MW: $p = 0.091$, $\Delta = 0.42$). While in the *Ghost* project, another *dependabotbot* shows signs of merge counts increasing as an immediate effect after the bot's adoption (lm: estimate = 6.12, $p = 0.079$). These situations make it very hard to infer if effects are positive or negative, making the results of the model very hard to interpret. While the results on code management bots suggest that they can significantly shape pull-request flows, their influence varies depending on project context.

Ultimately, it seems that activity duration by itself rarely altered project metrics, but longer lasting bots indicate that projects found their effects valuable enough to keep them. This of course applies better for bots that have been adopted earlier in the project, because bots that were adopted later may have had their activity duration shortened due to the end of our data collection periods.

Overall, although code and label management bots emerge as the most impactful in shaping project metrics, their effect is very project-specific. Commenting bots produced weaker and more variable effects. This is probably due to how varied the purpose of each of these bots actually is. State management bots largely did not have an impact on project metrics, although they help cleaning the workflow it could be that their work does not have a great impact that could help or disturb the developers. Across all bot types, the intensity of their activity rates strongly determined whether effects became visible. At the same time, much of the variance in project metrics were explained by developer count, suggesting that developer activity has a great impact on project metrics which cannot be ignored.

Answer to RQ3

Bot adoption does influence project metrics, but its effects are highly context-dependent and often overshadowed by broader project trends such as changes in developer participation. Code management and label management bots show the strongest and most consistent impacts, with code management bots altering merge rates and pull-request closing times, and label management bots increasing issue resolution counts while sometimes prolonging resolution times. By contrast, commenting and state management bots have weaker or more variable effects. Intense activity rates increase the likelihood of measurable changes, but multicollinearity in the models and overlapping bot adoptions make the interpretation difficult.

5 Threats to Validity

We now present the threats to validity, divided into internal and external threats.

5.1 Internal Validity

The correctness of our data relies on Codeface, GithubWrapper, and codeface-extraction as we use these tools to extract our data. And context and centrality metrics rely completely on Coronet as it is used for building our networks and reading Codeface data. Since these tools are well established and have been used in numerous previous studies [5, 6, 23], we assume they function as intended. The correctness of our bot detection relies on GitHub's internal bot labeling, BoDeGHa [20], and a manually curated subset by Wyrich et al. [44]. Since the dataset and these tools are well established, we assume that double-checking these bots through manual inspection of their GitHub profile as has been done by Bock et al. [6], gives us a reliable dataset. There are also some limitations to the way we approached the research questions. For example, the classification of bot roles is partly based on qualitative observations and interpretations, which introduces a degree of subjectivity. Different researchers might group bot behaviors slightly differently depending on how they interpret their activity. To reduce potential bias, the classification was grounded in actual issue event data from GitHub.

For RQ2, it can be critiqued that we rely on descriptive statistics over formal testing because it shows less rigor, but this aligns with the exploratory nature of the question. Additionally, when analyzing the impact of bots (RQ3), the study uses established and widely accepted metrics [10, 28, 40] (e.g., pull request resolution time, number of comments) and compares time periods before and after bot adoption, which helps to ensure some consistency and comparability with past work.

Although this helped us gather information on more bots, our grouped adoptions made it complicated to interpret data in RQ3. Combined with segmented regression models that induce multicolinearity, the analysis is less reliable. But thanks to the small scale of this study, we were able to derive a meaningful analysis, through the knowledge of the projects and supporting the models with Mann-Whitney U tests. We think it was important to consider these models, as it appeared that a lot of metrics external to bot adoption were inducing important significant effects. Throughout the thesis, we missed out on considering interesting activity metrics such as the median number of triggered events per human developer. This was mainly due to time constraints and our focus on avoiding overly complex models where additional correlated predictors (e.g., activity intensity per developer) could have introduced multicollinearity. While the metrics we used effectively captured

project activity, exploring additional metrics in future work (e.g., event distributions across contributors, ratios of bot- to human-triggered events, or activity per core developer) could provide further insights.

By looking almost exclusively at issue networks, we also have a limitation in core-developer identification for RQ2 as some core developers might show more centrality in co-change networks. But we believe that issue networks by themselves manage to illustrate the peripheral-core context meaningfully. Finally, previous work on bot influence have used hundreds to thousands of projects, whilst we are only considering 10 projects. While our sample size does not allow us to use complex time series models such as RDD, our selection of statistical tests should still enable us to capture possible changes in project behavior.

5.2 External Validity

This thesis focuses on a small number of OSS projects. While this allows for deep context-sensitive analysis, it also limits the extent as to how well the findings can be generalized to other OSS projects. Additionally, the project selection was not made randomly which may introduce selection bias, which also limits how well the case studies are representing all OSS projects. But the methodologies used draw on well-established metrics (e.g., centrality in developer networks, pull request resolution times), they use consistent time frames and rely on practices observed in other related works. Hence, the results can be compared with prior and possibly with future studies. Although generalization is limited by our project selection, the depth of our analysis provides strong groundwork for future large-scale research.

6 Concluding Remarks

6.1 Conclusion

In this thesis, we analyzed software bots across ten OSS projects to investigate how they influence project metrics, developer dynamics, and the interpretation of these dynamics. We attributed activity roles to bots by considering three aspects: the types of events they triggered, the duration of their contributions, and the rate at which they acted. Using developer issue-networks, we examined how bots fit into collaboration structures, while before–after adoption comparisons of project metrics allowed us to assess their measurable effects. Together, these analyses provided a comprehensive view of both the roles bots take on and their impacts on OSS projects.

Our main findings show that bot utilization varies widely across projects, not only in functionality but also in how long and how intensively they are employed. Regardless of this variation, bots consistently occupy highly central positions in developer networks, often with very stable eigenvector centrality scores. This suggests that once introduced, bots quickly integrate into the communication structure and remain influential. However, this influence comes at a cost: their presence can distort network interpretations by overshadowing the contributions of human developers, sometimes making humans who frequently interact with bots appear more central than they truly are. We also found that the rate and type of bot activity play decisive roles in shaping network centrality and influencing project metrics. Bots with intense activity rates tend to achieve higher centrality, as their frequent interactions link them strongly to other contributors. Additionally, activity types such as code management or label management are strongly associated with measurable improvements in pull request handling and issue resolution. In contrast, bots focused on commenting or state management exhibit weaker or inconsistent effects. Our results indicate that not all interactions carry the same weight: interactions that connect bots to maintainers or other influential actors are especially impactful.

Taken together, these findings suggest that bots are important actors in OSS development, shaping both collaboration networks and project outcomes. While this study is small in scale, the results are significant and point to clear links between bot activity roles, their centrality, and changes in project metrics. More importantly, this research shows the potential of looking closely at these aspects to better understand how bots affect OSS projects. By doing so, this thesis highlights the importance of careful analysis and offers new insights into the role of bots in OSS projects.

6.2 Future Work

This thesis has given a first look at how bot roles, activity patterns, and adoptions affect OSS projects, but there are several directions for future work. A natural next step would be to extend the study to a larger number of projects and longer time periods. This would make it possible to see whether the findings we observed here also appear in different OSS communities, of different sizes or domains.

Another step would be to refine our role classification. Our results could be combined with existing bot taxonomies (see Section 2.1), and the grouping of activity types and activity rates could be made more fine-grained. This could help identify subtler patterns in how different types of bots influence collaboration and project activity.

From a methodological point of view, there is room to improve how we measure changes in project metrics after bot adoption. In particular, using models that better handle multicollinearity while still capturing both immediate and long-term changes could strengthen the analysis and give clearer results.

Finally, our work suggests that bots can distort how developer networks are interpreted, since they often appear more central than humans or make maintainers look more central because of repeated interactions. Future research could test ways of filtering or adjusting bot activity in networks to reduce this distortion and give a more accurate picture of human collaboration. Another interesting direction is to apply these methods to GitHub's new AI agents[1], which may combine several activity types and play an even larger role in projects than the bots studied here.

1 https://github.com/resources/articles/ai/what-are-ai-agents/

A Appendix

A.1 Bot Activity for each Project

Table A.1: Bot Activity Overview for GOOGLE-DTP

Bot Name	Event Count	Activity Duration	Event Types
dependabotbot	36	11 Months	`commented` (12), `created` (11), `head_ref_deleted` (2), `labeled` (11)
googlebot	29	28 Months	`commented` (29)

Table A.2: Bot Activity Overview for REDUX

Bot Name	Event Count	Activity Duration	Event Types
gitbook-bot	1	Once	`commit_added`
netlifybot	746	26 Months	`commented`, `add_link`
snyk-bot	72	2 Days	`commit_added`, `commented`, `created`
ImgBotApp	1	Once	`commit_added`
github-actionsbot	7	7 Months	`commented`

Table A.3: Bot Activity Overview for NEXTJS

Bot Name	Event Count	Activity Duration	Event Types
greenkeeperbot	1499	16 Months	add_link (225), commented (231), commit_added (342), created (188), head_ref_deleted (92), labeled (181), referenced_by (235), state_updated (5)
greenkeeperio-bot	349	2 Months	add_link (57), commented (81), commit_added (81), created (81), referenced_by (49)
kodiakhqbot	4335	18 Months	commented (22), head_ref_deleted (49), merged (1013), referenced (84), referenced_by (1715), review_requested (6), state_updated (1446)
github-actionsbot	2446	5 Months	commented (2446)
lockbot	7154	25 Months	commented (249), locked (6894), unassigned (11)
Vercel	108	41 Months	comment_deleted (48), locked (59), unlocked (1)
vercelbot	1120	6 Months	add_link (5), commented (25), deployed (1086), mentioned (1), referenced_by (2), subscribed (1)
Coveralls	391	20 Days	add_link (261), commented (130)
codetriage-readme-bot	5	Once	commented (1), commit_added (1), created (1), mentioned (1), subscribed (1)
borsbot	1	Once	referenced (1)
dependabotbot	28	Once	commented (4), created (3), head_ref_deleted (1), head_ref_force_pushed (3), labeled (3), review_dismissed (1), review_requested (12), state_updated (1)

Table A.4: Bot Activity Overview for gocs

Bot Name	Event Count	Activity Duration	Event Types
dependabotbot	552	18 Months	add_link (5), commented (166), created (141), head_ref_deleted (21), head_ref_force_pushed (19), labeled (139), referenced_by (3), review_requested (47), state_updated (11)
dependabot-previewbot	434	15 Months	add_link (13), commented (135), created (102), head_ref_deleted (28), head_ref_force_pushed (25), labeled (98), referenced_by (10), review_requested (8), state_updated (15)
imgbotbot	6	2 Months	commented, created
Bot from GolangCI	30	2 Months	add_link (3), commented (16), referenced_by (3), reviewed (8)
github-actionsbot	5827	12 Months	commented (79), locked (5669), reviewed (79)
IssueHunt Bot	48	26 Months	commented (20), mentioned (18), subscribed (10)
CLAassistant	4	One Day	commented (4)
Codecov	11	11 Months	commented, add_link
fossabot	4	2 Days	commented (2), commit_added (1), created (1)
codecovbot	2180	37 Months	add_link (1304), commented (438), referenced_by (438)
ImgBotApp	3	2 Months	commit_added (3)
lgtm-combot	18	1 Month	add_link (12), commented (6)
Dependabot	2	18 Months	mentioned (2)
DeepSource Bot	26	3 Days	commented (12), commit_added (2), created (12)

Table A.5: Bot Activity Overview for BOOTSTRAP

Bot Name	Event Count	Activity Duration	Event Types
dependabotbot	955	7 Months	add_link (44), commented (177), commit_added (72), created (155), head_ref_deleted (5), head_ref_force_pushed (32), labeled (310), referenced_by (2), review_requested (154)
Bootstraps Cross-browser JS Tester Bot	631	29 Months	add_link (250), commented (285), mentioned (22), referenced (1), referenced_by (51), subscribed (22)
Bootstraps HTML Validator Bot	1824	28 Months	add_link (6), commented (608), mentioned (605), subscribed (605)
Bootstraps Issue Closer Bot	182	38 Months	commented (94), state_updated (88)
Bootstraps Pull Request Checker Bot	695	34 Months	commented (180), mentioned (168), renamed (1), state_updated (177), subscribed (168)
Coveralls	2	Once	add_link (1), commented (1)
Dependabot	4	1 Month	mentioned (4)
dependabot-previewbot	1615	8 Months	add_link (108), base_ref_changed (3), commented (283), commit_added (4), created (248), head_ref_deleted (17), head_ref_force_pushed (61), labeled (481), mentioned (1), referenced_by (11), review_requested (387), state_updated (11)
Travis Bot	1069	3 Months	add_link (642), commented (322), mentioned (1), subscribed (104)
fossabot	3	Once	commented (1), commit_added (1), created (1)
Hound	708	23 Months	add_link (1), commented (380), mentioned (3), reviewed (322), subscribed (2)
Odoos Mergebot	18	6 Months	referenced (18)
Odoo Forward-Port Steward bot	8	24 Days	referenced (8)
pullbot	5	27 Months	referenced_by (4), state_updated (1)
rpdesign	21	11 Months	add_link (2), commented (13), created (2), referenced_by (1), state_updated (3)
snyk-bot	1	Once	commit_added (1)

Table A.6: Bot Activity Overview for REACT

Bot Name	Event Count	Activity Duration	Event Types
borsbot	106	2 Months	referenced (106)
CodeSandbox	1	Once	mentioned (1)
codesandboxbot	3490	14 Months	add_link (1802), commented (1635), referenced_by (53)
Coveralls	106	4 Months	add_link (67), commented (39)
dependabotbot	222	16 Months	add_link (6), commented (90), commit_added (1), created (50), head_ref_deleted (25), labeled (50)
Facebook Community Bot	15559	80 Months	add_link (1), commented (2758), labeled (10803), mentioned (849), referenced (146), subscribed (849), unlabeled (153)
Imgbot	4	2 Months	commented (1), created (1), head_ref_deleted (1), state_updated (1)
openstack-gerrit	1	Once	referenced (1)
Pull Request Analyser	833	8 Months	add_link (549), commented (279), created (1), mentioned (2), subscribed (2)
pullbot	10	5 Months	referenced_by (10)
React Community Bot	525	30 Months	add_link (177), commented (169), labeled (151), unlabeled (28)
stalebot	2992	10 Months	commented (1355), labeled (826), state_updated (524), unassigned (1), unlabeled (286)

Table A.7: Bot Activity Overview for JQUERY

Bot Name	Event Count	Activity Duration	Event Types
lockbot	4384	24 Months	`locked` (4384)
jquerybot	814	52 Months	`add_link` (97), `commented` (113), `labeled` (502), `mentioned` (11), `subscribed` (11), `unlabeled` (80)
Mention Bot	1350	21 Months	`add_link` (1), `commented` (205), `mentioned` (572), `subscribed` (572)
Discourse Review Bot	1	Once	`referenced` (1)
JSF CLA Assistant	240	47 Months	`commented` (138), `mentioned` (51), `subscribed` (51)
Gulajava Ministudio	1	Once	`referenced_by` (1)
The Codacy Badger	3	Once	`commented` (1), `commit_added` (1), `created` (1)
pullbot	2	Once	`referenced` (2)

Table A.8: Bot Activity Overview for KERAS

Bot Name	Event Count	Activity Duration	Event Types
stalebot	8474	7 Months	`commented` (2154), `labeled` (3278), `state_updated` (2839), `unlabeled` (203)
Coveralls	2	Once	`add_link` (1), `commented` (1)
tensorflow-butlerbot	131	3 Months	`add_link` (4), `commented` (4), `labeled` (119), `referenced_by` (4)
TensorFlow Copybara	2	9 Months	`referenced` (2)

Table A.9: Bot Activity Overview for ghost

Bot Name	Event Count	Activity Duration	Event Types
greenkeeperio-bot	2218	15 Months	add_link (594), commented (356), commit_added (330), created (356), referenced_by (582)
Renovate Bot	30116	48 Months	add_link (568), commented (2209), commit_added (1579), created (2024), head_ref_deleted (881), head_ref_force_pushed (19971), head_ref_restored (133), merged (210), referenced_by (221), renamed (1355), review_requested (5), state_updated (960)
Codecov Comments Bot	937	2 Months	add_link (639), commented (149), referenced_by (149)
codecovbot	15524	18 Months	add_link (10652), commented (2437), referenced_by (2435)
Coveralls	17	Few Days	add_link (8), commented (9)
dependabotbot	5	Once	commented (2), created (1), head_ref_deleted (1), labeled (1)
EchoBilism	17	20 Days	add_link (1), commented (8), created (7), referenced_by (1)
Ghost CI	3374	22 Months	add_link (28), assigned (1236), commented (567), commit_added (8), labeled (576), mentioned (208), state_updated (175), subscribed (208), unlabeled (368)
Imgbot	2	Once	commented (1), created (1)
label-actionsbot	720	30 Months	commented (185), mentioned (177), state_updated (181), subscribed (177)
sentry-iobot	12	2 Months	assigned (4), commented (4), created (4)
snyk-bot	15	1 Week	commit_added (15)
stalebot	931	33 Months	commented (332), labeled (332), state_updated (178), unlabeled (89)

Table A.10: Bot Activity Overview for VUE

Bot Name	Event Count	Activity Duration	Event Types
Codecov	75	29 Months	add_link (45), commented (15), referenced_by (15)
codetriage-readme-bot	7	9 Months	commented (1), commit_added (1), created (1), head_ref_force_pushed (1), mentioned (1), state_updated (1), subscribed (1)
Coveralls	6	9 Months	add_link (4), commented (2)
dependabotbot	81	14 Months	add_link (1), commented (22), commit_added (3), created (19), head_ref_deleted (10), head_ref_force_pushed (5), labeled (19), referenced_by (1), state_updated (1)
Gulajava Ministudio	16	4 Months	referenced_by (16)
pullbot	1	Once	referenced_by (1)
vue-bot	2754	45 Months	add_link (1), commented (1280), commit_added (169), created (106), mentioned (19), referenced_by (1), state_updated (1159), subscribed (19)
vuejs-jp-bot	3	3 Months	referenced (3)

A.2 Roles for each Bot

Table A.11: Bot Activity Roles Overview (Part 1)

Bot Name	Category	Activity Type
Google-dtp		
dependabotbot	Extended Trial	Code Management
googlebot	Extended Trial	Commenting
Bootstrap		
dependabotbot	Short – Intense	Code Management
dependabot-previewbot	Short – Intense	Code Management
fossabot	Trial	Code Management
Bootstraps Cross-browser JS Tester Bot	Mid – Light	Commenting
Bootstraps HTML Validator Bot	Mid – Medium	Commenting
Bootstraps Issue Closer Bot	Long – Light	Commenting
Bootstraps Pull Request Checker Bot	Long – Light	Commenting
The Travis Bot	Short – Intense	Commenting
Hound	Mid – Medium	Commenting
Coveralls	Trial	Referenced or Mentioned
Dependabot	Trial	Referenced or Mentioned
Odoos Mergebot	Extended Trial	Referenced or Mentioned
Otis - Odoo Forward-Port Steward bot	Trial	Referenced or Mentioned
pullbot	Trial	Referenced or Mentioned
rpdesign	Extended Trial	Referenced or Mentioned
snyk-bot	Trial	Code Management

Table A.12: Bot Activity Roles Overview (Part 2)

Bot Name	Category	Activity Type
Ghost		
greenkeeperio-bot	Mid – Intense	Code Management
Renovate Bot	Long – Intense	Code Management
Ghost CI	Mid – Intense	Code Management
snyk-bot	Trial	Code Management
Codecov Comments Bot	Short – Intense	Commenting
codecovbot	Mid – Intense	Commenting
Coveralls	Trial	Commenting
dependabotbot	Trial	Commenting
EchoBilism	Trial	Commenting
ImgBot	Trial	Commenting
Label-actionsbot	Long – Light	Commenting
Sentry-iobot	Extended Trial	Commenting
stalebot	Long – Light	Label Management
Gogs		
ImgBotApp	Trial	Code Management
dependabotbot	Mid – Medium	Code Management
dependabot-previewbot	Mid – Light	Code Management
fossabot	Trial	Code Management
DeepSource Bot	Trial	Code Management
imgbotbot	Extended Trial	Commenting
GolangCI	Extended Trial	Commenting
IssueHunt Bot	Extended Trial	Commenting
CLAassistant	Trial	Commenting
Codecov	Extended Trial	Commenting
codecovbot	Long – Medium	Commenting
lgtm-combot	Extended Trial	Referenced or Mentioned
Dependabot	Trial	Referenced or Mentioned
DeepSource Bot	Trial	Referenced or Mentioned
github-actionsbot	Mid – Intense	State Management

Table A.13: Bot Activity Roles Overview (Part 3)

Bot Name	Category	Activity Type
JQUERY		
The Codacy Badger	Trial	Code Management
jquerybot	Mid – Medium	Commenting
Mention Bot	Long – Light	Commenting
JSF CLA Assistant	Long – Light	Commenting
Discourse Review Bot	Trial	Referenced or Mentioned
Gulajava Ministudio	Trial	Referenced or Mentioned
pullbot	Trial	Referenced or Mentioned
lockbot	Mid – Intense	State Management
KERAS		
Coveralls	Trial	Commenting
stalebot	Short – Intense	Label Management
Tensorflow-butlerbot	Short – Medium	Label Management
Tensorflow-Copybara	Trial	Referenced or Mentioned
NEXTJS		
greenkeeperbot	Mid – Medium	Code Management
kodiakhqbot	Mid – Intense	Code Management
codetriage-readme-bot	Trial	Code Management
github-actionsbot	Short – Intense	Commenting
vercelbot	Short – Intense	Commenting
Coveralls	Short – Intense	Commenting
borsbot	Trial	Referenced or Mentioned
dependabot	Trial	Referenced or Mentioned
lockbot	Mid – Intense	State Management
Vercel	Long – Light	State Management

Table A.14: Bot Activity Roles Overview (Part 4)

Bot Name	Category	Activity Type
React		
ImgBot	Trial	Code Management
codesandboxbot	Mid – Intense	Commenting
Coveralls	Short – Light	Commenting
Pull Request Analyser	Short – Medium	Commenting
dependabotbot	Mid – Light	Label Management
Facebook Community Bot	Long – Intense	Label Management
React Community Bot	Long – Light	Label Management
stalebot	Mid – Intense	Label Management
borsbot	Short – Medium	Referenced or Mentioned
Codesandbox	Trial	Referenced or Mentioned
Openstack-gerrit	Trial	Referenced or Mentioned
pullbot	Extended Trial	Referenced or Mentioned
Redux		
snyk-bot	Trial	Code Management
ImgBotApp	Trial	Code Management
netlifybot	Mid – Light	Commenting
github-actionsbot	Extended Trial	Commenting
Gitbook-bot	Trial	Referenced or Mentioned
Vue		
codetriage-readme-bot	Extended Trial	Code Management
dependabotbot	Mid – Light	Code Management
vue-bot	Long – Medium	Code Management
Codecov	Mid – Light	Commenting
Coveralls	Extended Trial	Commenting
Gulajava Ministudio	Extended Trial	Referenced or Mentioned
pullbot	Trial	Referenced or Mentioned
vuejs-jp-bot	Trial	Referenced or Mentioned

a.3 Significant Bot Adoption Effects

Table A.15: Significant effects of bots on project metrics captured by our linear models (Part 1)

Bot	Metric	Effect	Significance	Effect size / Direction
Redux				
Netlifybot	Res_count	Trend	•	Med (0.151) / Neg
Jquery				
JqueryBot	Merges	Level	*	Lg (0.264) / Pos
JqueryBot	Cmt_tot	Level	•	S (0.039) / Neg
JqueryBot	Res_count	Level	•	Med (0.201) / Pos
MentionBot	Res_time	Level	•	Med (0.138) / Neg
JSF CLA Assistant	Res_time	Level	•	Med (0.165) / Pos
Keras				
StaleBot	Res_count	Level	*	Lg (0.449) / Pos
StaleBot	Res_count	Trend	**	Lg (0.319) / Neg
StaleBot	Res_time	Level	*	Lg (0.461) / Pos
StaleBot	Res_time	Trend	*	Med (0.222) / Neg
Tensorflow-Butlerbot	Merges	Trend	**	Lg (0.339) / Pos
Tensorflow-Butlerbot	Rejects	Level	*	Med (0.228) / Neg
Tensorflow-Butlerbot	Cmt_tot	Trend	**	Lg (0.623) / Pos
Tensorflow-Butlerbot	Res_count	Trend	**	Lg (0.369) / Pos
Vue				
Codecov	Rejects	Level	•	S (0.036) / Neg
Codecov	PR_time	Trend	*	Med (0.200) / Pos
Codecov	Res_time	Trend	•	Med (0.147) / Pos
vue-bot	Res_count	Trend	*	Lg (0.254) / Neg
vue-bot	Res_time	Trend	*	Med (0.208) / Pos
dependabot	Merges	Level	•	Med (0.145) / Pos
dependabot	Merges	Trend	**	Lg (0.492) / Pos
dependabot	Rejects	Trend	**	Lg (0.351) / Pos
dependabot	PR_time	Trend	*	Lg (0.294) / Pos
dependabot	Res_time	Level	*	Med (0.188) / Pos
dependabot	Res_time	Trend	*	Lg (0.255) / Pos

Table elements are explained on the next page.

Table A.16: Significant effects of bots on project metrics captured by our linear models (Part 3)

Bot	Metric	Effect	Significance	Effect size / Direction
GOOGLE-DATA-TRANSFER-PROJECT				
dependabotbot	Cmt_tot	Trend	•	Med (0.172) / Pos
dependabotbot	Res_time	Trend	*	Lg (0.643) / Neg
GOGS				
Codecov / Dependabot	Merges	Level	*	Lg (0.351) / Pos
Codecov / Dependabot	Merges	Trend	*	Lg (0.303) / Neg
Codecov / Dependabot	Cmt_tot	Trend	**	Lg (0.443) / Neg
Codecov / Dependabot	Res_count	Trend	•	Med (0.145) / Neg
GithubActions	Rejects	Level	*	Lg (0.257) / Neg
GithubActions	PR_time	Trend	•	Med (0.178) / Pos
GithubActions	Res_count	Trend	*	Med (0.214) / Neg
NEXTJS				
Vercel	Res_count	Level	*	Lg (0.463) / Neg
Kodiakhbot	Rejects	Trend	•	Med (0.140) / Pos
Kodiakhbot	PR_time	Trend	*	Med (0.187) / Pos
Kodiakhbot	Cmt_tot	Trend	*	Lg (0.254) / Pos
Lockbot / Greenkeeper End	Merges	Trend	**	Lg (0.431) / Pos
Lockbot / Greenkeeper End	Cmt_tot	Trend	*	Lg (0.285) / Pos
vercelbot / end of lockbot	Merges	Trend	**	Lg (0.344) / Neg
vercelbot / end of lockbot	PR_time	Trend	•	Med (0.188) / Pos
vercelbot / end of lockbot	Cmt_tot	Trend	**	Lg (0.410) / Neg
BOOTSTRAP				
BootStrap Bots	Cmt/iss	Trend	*	Lg (0.258) / Neg
BootStrap Bots	Cmt_tot	Trend	•	Med (0.146) / Neg
Issue Closer / Hound	Res_count	Trend	**	Lg (0.359) / Pos
Dependabot	Merges	Level	•	Med (0.156) / Pos
Dependabot	Cmt/iss	Trend	**	Lg (0.339) / Pos
Dependabot	Res_count	Trend	•	Med (0.163) / Pos

Metric short names: Merges = merged PR count; Rejects = rejected PR count; PR_time = median PR closing time; Cmt/iss = median comments per issues; Cmt_tot = comments count; Res_count = non-PR issue resolutions count; Res_time = median issue resolution time.
Significance (the p-value of the effect): ** $p < 0.01$, * $p < 0.05$, • $0.05 \leq p < 0.10$.
Effect-size uses partial_R2 (Large / Medium / Small) shown in parentheses; direction is Positive/Negative/Mixed.

Table A.17: Significant effects of bots on project metrics captured by our linear models (Part 3)

Bot	Metric	Effect	Significance	Effect size / Direction
React				
Facebook Community Bot	Merges	Level	*	Lg (0.382) / Neg
Facebook Community Bot	PR_time	Trend	*	Lg (0.251) / Neg
React Community Bot	Merges	Trend	**	Lg (0.483) / Neg
React Community Bot	Rejects	Trend	**	Lg (0.305) / Pos
React Community Bot	PR_time	Trend	**	Lg (0.444) / Pos
React Community Bot	Cmt/iss	Trend	**	Lg (0.330) / Pos
Coveralls	Merges	Level	•	Med (0.106) / Pos
Coveralls	Merges	Trend	*	Med (0.189) / Pos
Coveralls	Cmt/iss	Level	**	Lg (0.305) / Neg
Coveralls	Res_count	Level	*	Med (0.240) / Neg
Pull Request Analyser / End of ReactCB	Res_time	Trend	•	Med (0.163) / Pos
Dependabot / Stalebot / codesandboxbot	Merges	Trend	**	Lg (0.299) / Neg
Dependabot / Stalebot / codesandboxbot	Res_count	Trend	•	Med (0.149) / Neg
Ghost				
Stalebot / Label-actionsbot / Renovate Bot	Merges	Trend	*	Med (0.239) / Neg
Stalebot / Label-actionsbot / Renovate Bot	PR_time	Trend	*	Med (0.212) / Pos
Stalebot / Label-actionsbot / Renovate Bot	Cmt_tot	Trend	•	Med (0.152) / Neg
Stalebot / Label-actionsbot / Renovate Bot	Res_count	Trend	**	Lg (0.329) / Neg
Stalebot / Label-actionsbot / Renovate Bot	Res_time	Trend	*	Med (0.239) / Neg
Ghost CI / Codecovbot	Cmt_tot	Trend	*	Med (0.185) / Pos

Metric short names: Merges = merged PR count; Rejects = rejected PR count; PR_time = median PR closing time; Cmt/iss = median comments per issues; Cmt_tot = comments count; Res_count = non-PR issue resolutions count; Res_time = median issue resolution time.
Significance (the p-value of the effect): ** $p < 0.01$, * $p < 0.05$, • $0.05 \leq p < 0.10$.
Effect-size uses partial_R2 (Large / Medium / Small) shown in parentheses; direction is Positive/Negative/Mixed.

a.4 Significant Mann-Whitney U Results (low p-value and medium/large effect size)

Table A.18: Significant Mann-Whitney U test results for bots on project metrics (Part 1)

Bot	Metric	W	Significance	Cliff's delta
Redux				
Netlifybot	Res_count	17	**	-0.76
Jquery				
JqueryBot	Merges	121	**	0.69
JqueryBot	PR_time	113	*	0.57
JqueryBot	Cmt/iss	106	*	0.48
JqueryBot	Cmt_tot	124	**	0.72
JqueryBot	Res_count	133	**	0.85
MentionBot	Merges	36	*	-0.50
MentionBot	Rejects	26	**	-0.63
MentionBot	PR_time	29	*	-0.60
MentionBot	Cmt/iss	38	*	-0.47
MentionBot	Res_time	32	*	-0.56
JSF CLA Assistant	Merges	106	*	0.47
JSF CLA Assistant	Rejects	28	*	-0.60
JSF CLA Assistant	Cmt/iss	27	**	-0.62
JSF CLA Assistant	Cmt_tot	25	**	-0.65
JSF CLA Assistant	Res_count	28	*	-0.61
LockBot	Rejects	32	*	-0.55
LockBot	Res_time	118	**	0.64
Keras				
StaleBot	Cmt_tot	26	**	-0.64
StaleBot	Res_time	114	*	0.58
Tensorflow-Butlerbot	Merges	2	**	-0.97
Tensorflow-Butlerbot	Rejects	1	**	-0.99
Tensorflow-Butlerbot	PR_time	8	•	-0.67
Tensorflow-Butlerbot	Res_count	0	**	-1.00
Vue				
Codecov	Merges	111	*	0.54
Codecov	Rejects	108	*	0.50
Codecov	PR_time	109	*	0.51
Codecov	Cmt_tot	136	**	0.89
Codecov	Res_count	144	**	1.00
Codecov	Res_time	11	**	-0.85
vue-bot	PR_time	125	**	0.74
vue-bot	Cmt/iss	42	*	-0.42
vue-bot	Res_time	23	**	-0.68
dependabot	Merges	28	*	-0.61
dependabot	Cmt_tot	1	**	-0.99
dependabot	Res_count	1	**	-0.99

Table A.19: Significant Mann-Whitney U test results for bots on project metrics (Part 2)

Bot	Metric	W	Significance	Cliff's delta
Google-data-transfer-project				
dependabotbot	Cmt/iss	42	•	-0.41
dependabotbot	Cmt_tot	28	*	-0.60
dependabotbot	Res_count	41	•	-0.43
Gogs				
Codecov / Dependabot	Merges	102	•	0.42
Codecov / Dependabot	Rejects	104	•	0.45
Codecov / Dependabot	Cmt/iss	31	**	-0.56
GithubActions	Merges	116	*	0.61
GithubActions	Cmt/iss	27	*	-0.58
Nextjs				
Vercel	Merges	38	•	-0.47
Vercel	Cmt/iss	32	*	-0.47
Vercel	Cmt_tot	30	*	-0.58
Kodiakhbot	Merges	143	**	0.99
Kodiakhbot	Rejects	124	**	0.73
Kodiakhbot	Cmt/iss	118	**	0.64
Kodiakhbot	Cmt_tot	144	**	1.00
Kodiakhbot	Res_count	120	**	0.67
Lockbot / Greenkeeper End	Merges	115	*	0.60
Lockbot / Greenkeeper End	PR_time	36	*	-0.50
Lockbot / Greenkeeper End	Cmt/iss	36	*	-0.50
vercelbot / end of lockbot	Merges	36	*	-0.49
vercelbot / end of lockbot	PR_time	73	**	0.74
vercelbot / end of lockbot	Res_time	65	•	0.55
Bootstrap				
BootStrap Bots	Merges	25	**	-0.65
BootStrap Bots	Rejects	24	**	-0.67
BootStrap Bots	Cmt_tot	20	**	-0.72
BootStrap Bots	Res_count	11	**	-0.85
Issue Closer / Hound	Cmt/iss	48	*	-0.33
Issue Closer / Hound	Cmt_tot	40	•	-0.44
Issue Closer / Hound	Res_time	117	**	0.62
Dependabot	Rejects	102	•	0.42
Dependabot	PR_time	12	**	-0.83
Dependabot	Res_count	41	•	-0.43
Dependabot	Res_time	105	•	0.46

Metric short names: Merges = merged PR count; Rejects = rejected PR count; PR_time = median PR closing time; Cmt/iss = median comments per issues; Cmt_tot = comments count; Res_count = non-PR issue resolutions count; Res_time = median issue resolution time.
Significance (the p-value of the effect): ** $p < 0.01$, * $p < 0.05$, • $0.05 \leq p < 0.10$.
Cliff's delta is shown for medium/large effects.

Table A.20: Significant Mann-Whitney U test results for bots on project metrics (Part 3)

Bot	Metric	W	Significance	Cliff's delta
React				
Facebook Community Bot	Rejects	101	•	0.40
Facebook Community Bot	Cmt/iss	91	•	0.38
Facebook Community Bot	Cmt_tot	125	**	0.74
Facebook Community Bot	Res_count	122	**	0.70
React Community Bot	Merges	101	•	0.41
React Community Bot	Rejects	106	•	0.47
React Community Bot	PR_time	38	•	-0.47
React Community Bot	Cmt_tot	101	•	0.40
React Community Bot	Res_count	124	**	0.72
React Community Bot	Res_time	14	**	-0.81
Coveralls	Merges	101	•	0.41
Coveralls	Res_count	14	**	-0.80
Pull Request Analyser / End of ReactCB	Merges	18	**	-0.75
Pull Request Analyser / End of ReactCB	Rejects	32	*	-0.55
Pull Request Analyser / End of ReactCB	Cmt_tot	33	*	-0.54
Dependabot / Stalebot / codesandboxbot	Cmt/iss	101	•	0.40
Dependabot / Stalebot / codesandboxbot	Res_time	102	•	0.42
Ghost				
Greenkeeperio	Rejects	104	•	0.45
Greenkeeperio	Cmt_tot	35	*	-0.51
Greenkeeperio	Res_count	34	*	-0.52
Stalebot / Label-actionsbot / Renovate Bot	Merges	104	•	0.44
Stalebot / Label-actionsbot / Renovate Bot	PR_time	21	**	-0.71
Stalebot / Label-actionsbot / Renovate Bot	Cmt/iss	15	**	-0.78
Stalebot / Label-actionsbot / Renovate Bot	Cmt_tot	27	**	-0.62
Ghost CI / Codecovbot	Merges	125	**	0.74
Ghost CI / Codecovbot	Rejects	108	*	0.51
Ghost CI / Codecovbot	Cmt_tot	117	*	0.62

Metric short names: Merges = merged PR count; Rejects = rejected PR count; PR_time = median PR closing time; Cmt/iss = median comments per issues; Cmt_tot = comments count; Res_count = non-PR issue resolutions count; Res_time = median issue resolution time.
Significance (the p-value of the effect): ** $p < 0.01$, * $p < 0.05$, • $0.05 \leq p < 0.10$.
Cliff's delta is shown for medium/large effects.

Statement on the Usage of Generative Digital Assistants

For this thesis, the following generative digital assistants have been used: We have used ChatGPT-4o (OpenAI, version released May 2024) and ChatGPT-5 (OpenAI, version released August 2025)[1] for exploring alternative phrasing or structuring ideas in certain parts of the thesis. We are aware of the potential dangers of using these tools and have used them sensibly with caution and with critical thinking.

In particular, it was used to help articulate and organize notes and reflections based on previously gathered materials and papers, particularly in the section discussing background and related work. The assistants supported the phrasing and structure of these ideas, but all the content was originally developed by the authors and the final text was created using own phrasing. The assistants were also used as tools to debug parts of the R code used for this paper, helping to understand and resolve certain code errors. Additionally, they were used to convert Markdown tables from the authors' notes into LaTeX booktabs tables, specifically the tables presented in the Appendix.

1 Both version are available at: https://openai.com/index/chatgpt/

Bibliography

[1] 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 19th International Conference on Mining Software Repositories*. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, 6–17.

[2] J.M. Belmont. *Hands-On Continuous Integration and Delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI*. Packt Publishing, 2018.

[3] James Lopez Bernal, Steven Cummins, and Antonio Gasparrini. "Interrupted time series regression for the evaluation of public health interventions: a tutorial." In: *International Journal of Epidemiology* 46.1 (June 2016), pp. 348–355.

[4] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. "Mining email social networks." In: *Proceedings of the 2006 international workshop on Mining software repositories*. 2006, pp. 137–143.

[5] Thomas Bock. "Emerging organizational patterns in evolving open-source software projects: an analysis of developer activity and coordination." PhD thesis. 2024.

[6] Thomas Bock, Nils Alznauer, Mitchell Joblin, and Sven Apel. "Automatic Core-Developer Identification on GitHub: A Validation Study." In: *ACM Trans. Softw. Eng. Methodol.* 32.6 (Sept. 2023).

[7] Ulrik Brandes and Thomas Erlebach. *Network analysis: methodological foundations*. Vol. 3418. Springer Science & Business Media, 2005.

[8] Chris Brown and Chris Parnin. "Sorry to Bother You: Designing Bots for Effective Recommendations." In: *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 2019, pp. 54–58.

[9] Brown C. and Parnin C. "Sorry to Bother You: Designing Bots for Effective Recommendations." In: *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)* (2019), pp. 54–58.

[10] Nathan Cassee, Bogdan Vasilescu, and Alexander Serebrenik. "The Silent Helper: The Impact of Continuous Integration on Code Reviews." In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020, pp. 423–434.

[11] Natarajan Chidambaram, Alexandre Decan, and Tom Mens. "A Dataset of Bot and Human Activities in GitHub." In: *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. 2023, pp. 465–469.

[12] Natarajan Chidambaram, Alexandre Decan, and Tom Mens. "A bot identification model and tool based on GitHub activity sequences." In: *Journal of Systems and Software* 221 (2025).

[13] Kevin Crowston, Qing Li, Kangning Wei, U. Eseryel, and James Howison. "Self-organization of teams for free/libre open source software development." In: *Information and Software Technology* 49 (June 2007), pp. 564–575.

[14] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. "Core and periphery in free/libre and open source software team communications." In: *Proceedings of the 39th annual Hawaii international conference on system sciences (HICSS'06)*. Vol. 6. IEEE. 2006, 118a–118a.

[15] Subhajit Datta, Vikrant Kaulgud, Vibhu Saujanya Sharma, and Nishant Kumar. "A social network based study of software team dynamics." In: *Proceedings of the 3rd India software engineering conference*. 2010, pp. 33–42.

[16] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. "Detecting and Characterizing Bots that Commit Code." In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, 209–219.

[17] Ikram El Asri, Noureddine Kerzazi, Lamia Benhiba, and Mohammed Janati. "From periphery to core: a temporal analysis of github contributors' collaboration network." In: *Working Conference on Virtual Enterprises*. Springer. 2017, pp. 217–229.

[18] Linda Erlenhov, Francisco Gomes de Oliveira Neto, and Philipp Leitner. "An empirical study of bots in software development: characteristics and challenges from a practitioner's perspective." In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: ACM, 2020, 445–455.

[19] Harald Gall, Karin Hajek, and Mehdi Jazayeri. "Detection of logical coupling based on product release history." In: *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE. 1998, pp. 190–198.

[20] 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." In: *Journal of Systems and Software* 175 (2021), p. 110911.

[21] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. "Work practices and challenges in pull-based development: the contributor's perspective." In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16. Austin, Texas: Association for Computing Machinery, 2016, 285–296.

[22] Felipe Grando, Diego Noble, and Luis C Lamb. "An analysis of centrality measures for complex and social networks." In: *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2016, pp. 1–6.

[23] Christian Hechtl, Mitchell Joblin, and Sven Apel. "Is Perceived Gender Related to Contributions and Standing in Open-Source Software Projects?" In: *Empirical Software Engineering (EMSE)* (2025).

[24] Qiaona Hong, Sunghun Kim, Shing Chi Cheung, and Christian Bird. "Understanding a developer social network and its evolution." In: *2011 27th IEEE international conference on software maintenance (ICSM)*. IEEE. 2011, pp. 323–332.

[25] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. "Classifying developers into core and peripheral: an empirical study on count and network metrics." In: *Proceedings of the 39th International Conference on Software Engineering*. ICSE '17. Buenos Aires, Argentina: IEEE Press, 2017, 164–174.

[26] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. "From developer networks to verified communities: A fine-grained approach." In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE. 2015, pp. 563–573.

[27] 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." In: *ACM Trans. Softw. Eng. Methodol.* 33.2 (Dec. 2023).

[28] Timothy Kinsman, Mairieli Wessel, Marco A. Gerosa, and Christoph Treude. "How Do Software Developers Use GitHub Actions to Automate Their Workflows?" In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, May 2021, 420–431.

[29] Samim Mirhosseini and Chris Parnin. "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. ASE '17. Urbana-Champaign, IL, USA: IEEE Press, 2017, 84–94.

[30] Martínez-Palacios C.S. Ocharán-Hernández J.O. et al Moguel-Sánchez R. "Bots in Software Development: A Systematic Literature Review and Thematic Analysis." In: 49 (2023), pp. 712–734.

[31] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. "Evolution patterns of open-source software systems and communities." In: *Proceedings of the international workshop on Principles of software evolution*. 2002, pp. 76–85.

[32] S Santhanam, T Hecking, A Schreiber, and S. Wagner. "Bots in software engineering: a systematic mapping study." In: (2022).

[33] Jeffrey C. Johnson Stephen P. Borgatti Martin G. Everett. "Analyzing Social Network." In: *Sociological Methodology* 22 (1992), pp. 1–35.

[34] Margaret-Anne Storey, Alexander Serebrenik, Carolyn Penstein Rosé, Thomas Zimmermann, and James D. Herbsleb. "BOTse: Bots in Software Engineering (Dagstuhl Seminar 19471)." In: *Dagstuhl Reports* 9.11 (2020). Ed. by Margaret-Anne Storey, Alexander Serebrenik, Carolyn Penstein Rosé, Thomas Zimmermann, and James D. Herbsleb, pp. 84–96.

[35] Margaret-Anne Storey and Alexey Zagalsky. "Disrupting developer productivity one bot at a time." In: FSE 2016. Seattle, WA, USA: ACM, 2016, 928–931.

[36] Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. "Unveiling Elite Developers' Activities in Open Source Projects." In: *ACM Trans. Softw. Eng. Methodol.* 29.3 (June 2020).

[37] Mairieli Wessel. "Enhancing developers' support on pull requests activities with software bots." In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: Association for Computing Machinery, 2020, 1674–1677.

[38] Mairieli Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco A. Gerosa, and Igor Steinmacher. "Bots for pull requests: the good, the bad, and the promising." In: ICSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 274–286.

[39] Mairieli Wessel, Marco Aurelio Gerosa, and Emad Shihab. "Software bots in software engineering: benefits and challenges." In: *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, Oct. 2022, 724–725.

[40] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A. Gerosa. "Quality gatekeepers: investigating the effects of code review bots on pull request activities." In: Springer Science and Business Media LLC, 2022.

[41] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. "The Power of Bots: Characterizing and Understanding Bots in OSS Projects." In: *Proc. ACM Hum.-Comput. Interact.* 2.CSCW (Nov. 2018).

[42] 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'20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, 51–55.

[43] Mairieli Wessel, Joseph Vargovich, Marco A. Gerosa, and Christoph Treude. "GitHub Actions: The Impact on the Pull Request Process." In: Springer Science and Business Media LLC, 2023.

[44] Marvin Wyrich, Raoul Ghit, Tobias Haller, and Christian Müller. "Bots Don't Mind Waiting, Do They? Comparing the Interaction With Automatically and Manually Created Pull Requests." In: *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. 2021, pp. 6–10.

[45] 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 '17. Urbana-Champaign, IL, USA: IEEE Press, 2017, 60–71.