Bachelor's Thesis

UNCOVERING ACTIVITIES OF PRIVILEGED DEVELOPERS IN OPEN-SOURCE SOFTWARE PROJECTS

LEONIE SCHNEIDER

April 9, 2024

Advisor:
Thomas Bock Chair of Software Engineering
Christian Hechtl Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering
Prof. Dr. Ingmar Weber Chair of Societal Computing

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT DES SAARLANDES

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

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

Declaration of Consent

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

In open-source software (OSS) projects, certain developers have more privileges than others. These privileges allow developers to merge commits or lock issues, for example.
Compared to larger closed-source software projects, where developers have defined roles and the management department is responsible for making decisions, OSS projects typically lack a strict, predefined organizational hierarchy. The collaborative, decentralized nature of these projects presents a challenge when it comes to selecting appropriate candidates to receive privileges. Determining who qualifies as a valid candidate for privileges requires careful consideration, thus criteria should be established to simplify with the identification process.
This work is aiming to gain insights into the behavior of developers to find characteristics that distinguish future privileged developers from unprivileged developers. We analyze the issue and commit data from 9 different OSS projects hosted on GitHub and build developer networks from these data to obtain centrality metrics of developers. Using these data and metrics, we compare the behavior of developers who are later granted privileges with that of developers who are not granted privileges, and we investigate whether and in which way the behavior of developers changes after being granted privileges.
Our analyses reveal that future privileged developers are significantly more active than developers who do not receive privileges, evidenced by a higher number of commits and a higher number of overall triggered issue events. We also find that the change in behavior after receiving privileges depends on project size. In large projects with a high number of privileged developers, commits, and issues, developers become more active after receiving privileges and also become more centralized in the developer network.
The observed differences between future privileged and unprivileged developers suggest that one of the main characteristics of suitable developers for privileges is high engagement, resulting in a high number of commits and issue events. Since developers with these characteristics become more active after receiving privileges (at least, in large projects), we can suggest that identifying appropriate candidates based on these attributes is a suitable approach for finding promising developers for privileges.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Open-source software (OSS) projects are essential to modern software development. These projects have a publicly available code base and rely on voluntary contributions from a globally distributed developer community [26]. In closed-source software projects, centralized authorities define strict hierarchical structures among developers [13, 51]. In comparison, OSS projects are not formally organized and do not have a predefined command and control structure [3]. Developers typically organize themselves in a decentralized manner [3, 13, 51], hence hierarchical group structures also emerge in OSS projects [4, 43, 50, 52].
Since developers in OSS projects are distributed globally, they usually do not know each other personally [2, 4, 23]. Good coordination between them is therefore of great importance to ensure the quality and success of a project [4, 11, 24, 32, 33].

Research has shown that highly successful OSS projects often adopt a hybrid organizational structure that consists of a hierarchical part and a non-hierarchical part [29]. Even in the absence of predefined hierarchical group structures, certain developers have specific responsibilities in the project, including tasks related to maintenance and the implementation of core functionalities [4]. This leads to the establishment of different roles. Highly engaged developers, often called *maintainers* or *core developers*, play a central role in code contributions and project evolution [14, 41] and occupy higher positions in the hierarchy [30]. The remaining developers, often called *peripheral developers* [28], occupy lower positions in the hierarchy [30], as they are only involved on an irregular or short-term basis.

Given these different levels of involvement and expertise, not all developers should have the same privileges within the project. Developers with privileges, i.e., core developers, have significant influence on the direction of the project [1, 28, 45, 53] through actions like merging code into the code base or blocking users. Thus, careful consideration of the possible candidates for granting privileges is essential to ensure the long-term success of OSS projects. However, the absence of predefined hierarchical structures [3] and centralized authority [13, 51] presents challenges in identifying suitable candidates.

1.1 GOAL OF THIS THESIS

This study aims to address this problem by finding measures to identify suitable candidates to receive privileges in OSS projects. We attempt to gain insight into developer behavior to discover differences between future privileged and unprivileged developers. We obtain the number of commits and triggered issue events (e.g., `commented`, `commit_added`) per developer from 9 OSS projects on GitHub. Then, we compare the values of future privileged developers to those of unprivileged developers.
Further, we investigate how developer behavior changes after receiving privileges, by searching for changes in the number of commit and issue events, as well as changes in the

developer's centrality in developer networks. These analyses provide valuable insights, such as how developers adapt to new roles and responsibilities, and whether they frequently help other, less experienced developers. This, in turn, sheds light on whether the observed behavioral differences between future privileged and unprivileged developers are a reliable indicator of a developer's suitability for privileges. For example, if developers generally become less engaged after being granted privileges, using their pre-privilege behavior as a benchmark may not be appropriate.

Our results indicate that there is indeed a difference between future privileged and unprivileged developers. The former are significantly more active, contributing to more commits and triggering more issue events than the latter. Furthermore, once they are privileged, their activity and centrality within the project increases, at least in those projects with a high number of privileged developers, commits, and issues. Based on these results, we are able to identify characteristics of future privileged developers. These characteristics can be used to help in the process of finding suitable candidates to be given privileges in OSS projects.

1.2 OVERVIEW

In the following, we give an overview of the structure of this work.

In **Chapter** 2, we provide a comprehensive overview of the essential background information relevant to our study.

In **Chapter** 3, we then present our research questions and the methodology we use to address them.

In **Chapter** 4, we present our findings and discuss their implications in depth. Afterward, we outline potential threats to the validity of our study.

In **Chapter** 5, we give an overview of relevant prior work and place our work in this context.

Finally, in **Chapter** 6, we summarize our findings, draw conclusions, and suggest opportunities for future research.

2

BACKGROUND

This chapter provides fundamental background information for this thesis. We give an overview of the concept of OSS projects and GitHub, in general, and we break down how the various GitHub events relate to developer roles. Finally, we explain developer networks and the centrality metric we use.

2.1 OSS PROJECTS AND GITHUB

In software projects, developers need to be coordinated to ensure high-quality code and drive project evolution [11, 24, 32, 33]. However, the coordination of OSS projects is fundamentally different from that of commercial, closed-source software projects. OSS projects lack a clear, mandated organizational structure and a centralized authority that can establish leadership positions [4, 43, 50].

In addition, there is no clear separation between users and developers, since any user can be a developer [50]. Nevertheless, "hierarchical group structures among developers do exist" [4], but they evolve according to principles of self-organization [13, 51].

A commonly used platform for OSS development is GitHub[1], which is based on the version control system Git[2]. In the context of our research, GitHub provides a rich source of data for studying the behavior and activities of developers in OSS projects. In particular, two types of data available on GitHub are of special interest to our research: issue data and commit data.

GitHub issues serve as a mechanism for tracking tasks, reporting bugs, and submitting feature requests within projects [20]. Commits, on the other hand, represent changes made to the project's source code and provide a detailed view of development activities, such as which developer changed which part of the code [17]. Commits are often linked to specific issues to clarify the connection between tasks and the changes made to the code.

One way to integrate the changes from a commit into the project is for developers to create a pull request containing the commits they intend to merge [21]. There are several ways to merge these pull requests into the project, including automatic merging via GitHub [18], merging done by bots, or manual merging done by developers. In projects using the manual approach, only developers with additional permissions are able to merge pull requests [4]. A more detailed introduction to permissions on GitHub is given in Section 2.3.

Since pull requests often work on fixing bugs or implementing features, GitHub allows a pull request to be linked to an issue [20]. This helps developers see that someone is working

1 https://github.com (accessed 2024-03-15)
2 https://git-scm.com (accessed 2024-03-15)

on the issue. Internally, within the GitHub system, pull requests are treated as a special form of issues [4]. Therefore, whenever we talk about issues, this always also includes pull requests.

2.2 DEVELOPER ROLES

Not all developers contribute equally to OSS projects. Some developers take on specific responsibilities in the project [4], while others participate more irregularly [29]. This leads to the establishment of different groups of developers.

In the literature, there are different approaches to these groups. One of them is that developers can have different roles, being *project leader, core member, active developer, peripheral developer, bug fixer, bug reporter, reader,* and *passive user* [50]. Whereas project leaders are the initiators of the project who are responsible for the overall direction, and core members "are responsible for guiding and coordinating the development of an OSS project" [50], peripheral developers only "occasionally contribute new functionality or features to the existing system" [50], and bug reporters might not even read the source code and only discover and report bugs.

Another approach is to classify developers into only two different groups. On the one hand, there are the *maintainers*, or *core developers* [27, 28, 35, 38], who take responsibility for specific tasks in the project and become heavily involved, and on the other hand, there are the *peripheral developers* [12, 28, 35, 38], who contribute more sporadically.
Core developers possess in-depth knowledge of the project's source code and software architecture [41]. They actively participate in the development of the project [12, 35], hold decision-making roles [4, 41], and have access to sensitive aspects of the project, e.g., blocking users [4].
Peripheral developers are less consistently involved, contributing only occasionally [12, 14, 28, 29, 35, 44]. While they may assist with bug fixes or occasional feature additions, their contributions are not continuous enough to classify them as core developers. This group also includes one-time contributors, often referred to as newcomers [34, 40, 42].

Since many OSS projects do not provide a public list of core developers, there are several approaches to identify them.
A state-of-the-art approach suggests, for example, that developers with a high number of contributions should be considered core developers, since a minority of developers typically account for about 80 % of the source code contributions [35]. Using this approach, Crowston et al. [14] showed that the top 20 % of developers are responsible for 80 % of the commits. Canedo et al. [10] did not use the number of commits, but instead stated that a core developer must have authored, at least, 50 % of the files in the project.
The approach Bock et al. [4] chose is using the "privileged events in GitHub issue discussions and pull requests" [4], which is introduced in the following Section 2.3.

2.3 GitHub Events and Permissions

To define the criteria to identify which group a developer belongs to, we first need to look at GitHub issue events[3], since we use these events for the definition. As mentioned in Section 2.1, GitHub issues are used to track tasks. Issue events [19] represent various actions or activities that occur within those issues, some examples of different types of events are:

- `commented`: This event occurs when a developer adds a comment to an issue or pull request.
- `commit_added`: This event occurs when a commit is added to the pull request's HEAD branch.
- `mentioned`: This event occurs when a user is mentioned in an issue or pull request.

These issue events also provide us insights into the behavior of developers, as we can, for instance, see which specific events were triggered more or less frequently.

Having established the issue events, we can now define the criteria for classifying developers into their respective groups. There are various methods to identify which group a developer belongs to (core or peripheral), as detailed in Section 2.2.
Bock et al. [4] use the different permissions[4] GitHub grants to users at the project level, which are: *read, triage, write, maintain*, and *admin* permissions. The permissions stack on top of each other, meaning that any developer who has, e.g., *write* permission, also implicitly has *read* and *triage* permission [4].

Using these permissions, they classified the occurring events into 3 categories:

- **Common events**, which can be triggered by anyone (e.g., writing a comment).
- **Extended events**, which can only be triggered by users with, at least, triage permission (e.g., applying a label).
- **Privileged events**, which can only be triggered with, at least, write permission (e.g., merging a pull request).

The exact mapping of GitHub events to role permissions can be found in Table 2.1.

Any user who has triggered a privileged event is considered a core developer, since Bock et al. [4] assumed that an event that requires, at least, write permission, is either critical for the maintenance of the project or for decision making, both of which distinguish a core developer. The remaining users are considered peripheral developers [4].

We use their concept of core and peripheral developers and the 3 categories of events (common, extended, and privileged events) to figure out which developer belongs to which group, which then allows us to analyze the differences in behavior between these two groups.
Since core developers are defined as having triggered a privileged event, we refer to them as

3 https://docs.github.com/en/rest/using-the-rest-api/issue-event-types (accessed 2024-03-15)
4 https://docs.github.com/en/github/setting-up-and-managing-organizations-and-teams/repository-permission-levels-for-an-organization#repository-access-for-each-permission-level (accessed 2023-08-26)

Table 2.1: GitHub issue events and the role permissions needed to be able to trigger them, adapted from Bock et al. [4]

Privileged Events (write, maintain, or admin permission)	Extended Events (at least, triage permission)	Common Events (at least, read permission)
`added_to_project`, `converted_note_to_issue`, `deployed`, `deployment_environment_changed`, `locked`, `merged`, `moved_columns_in_project`, `pinned`, `removed_from_project`, `review_dismissed`, `transferred`, `unlocked`, `unpinned`, `user_blocked`	`assigned`, `demilestoned`, `labeled`, `marked_as_duplicate`, `milestoned`, `resolution_updated`, `type_updated` `unassigned`, `unlabeled`, `unmarked_as_duplicate`	`add_link`, `automatic_base_change_failed`, `automatic_base_change_succeeded`, `base_ref_changed`, `base_ref_force_pushed`, `comment_deleted`, `commented`, `commit_added`, `connected`, `convert_to_draft`, `created`, `cross_referenced`, `disconnected`, `head_ref_deleted`, `head_ref_force_pushed`, `head_ref_restored`, `mentioned`, `ready_for_review`, `referenced`, `referenced_by`, `renamed`, `review_request_removed`, `review_requested`, `reviewed`, `state_updated`, `subscribed`, `unsubscribed`

developers with privileges. Peripheral developers who only trigger common and extended events are referred to as non-privileged developers.

2.4 NETWORKS AND CENTRALITY

To analyze developer behavior, we also use networks in our research. Networks are data structures that are typically used to visualize relationships between many parties [46]. We use socio-technical developer networks, a special kind of network that is constructed based on the contributions of developers to the software. Vertices represent developers and edges their interactions with other developers [28].
We distinguish between two types of networks that differ in the type of connections between developers [4] (represented in Figure 2.1), to capture both collaborative and communicative aspects of developer behavior:

- **Developer Collaboration Networks:** The foundation of these networks is Git commit data. Developers are connected by an edge if they contributed changes to the same artifact, e.g., "when they have edited the same file within the same time window" [4]. These networks are also called *cochange* networks.
- **Developer Communication Networks:** The foundation of these networks is communication data, i.e., GitHub issue data. Two developers are connected by an edge "when they have contributed to the same issue or pull request within the same time window" [4]. These networks are also called *issue* networks.

We are particularly interested in the centrality of developers in these networks. Centrality refers to the importance of a developer within the network and can be measured using various centrality metrics [46]. One metric we can obtain from developer networks is *degree*

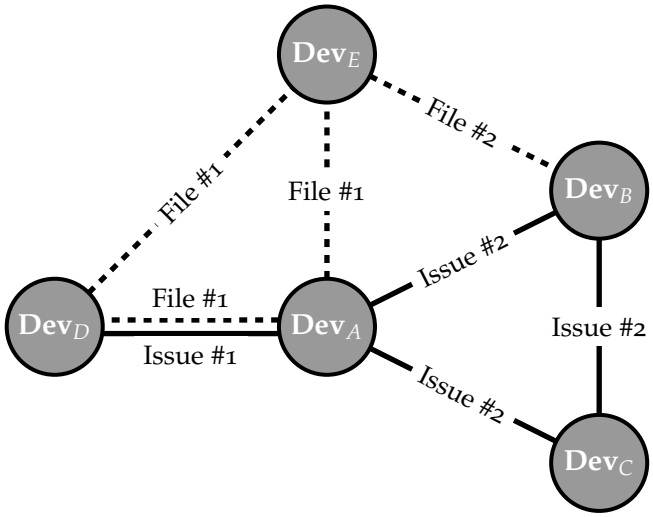

Figure 2.1: Both types of developer networks. Dashed lines represent edges in the cochange network, while solid lines represent edges in the issue network. The edge labels specify the file or issue that both developers worked on together.

centrality, which represents the number of edges a developer has to other developers [8]. Another centrality metric is *hierarchy degree*, which represents the position of a developer within the hierarchical structure of a network [28]. The centrality metric we use is *eigenvector centrality*, which recursively determines the centrality of a developer by the centrality of its neighboring vertices [6]. This metric can therefore be used to "weight the importance of developers by the importance of others they are interacting with" [4]. High eigenvector centrality in developers is observed either when they are linked to a large number of other developers or when their connections include developers who themselves have high eigenvector centrality [5–8].

3

METHODOLOGY

This section introduces the research questions we identify to analyze the behavior of privileged developers in OSS projects. We then outline our expected findings and explain our data preparation process. Lastly, we provide details of our analysis methods, including the statistical tests we use to answer the research questions.

3.1 RESEARCH QUESTIONS

The goal of our work is to analyze the differences between privileged and unprivileged developers. We aim at analyzing the behavior of both groups of developers and to understand the differences between them to obtain metrics to identify potential candidates in OSS projects who are suitable to be granted privileges. To achieve this, we identify 3 research questions.

3.1.1 *RQ1: Privileged vs. Unprivileged Behavior*

First, we are interested in whether there are any noticeable differences between the behavior of developers without privileges who receive privileges and the behavior of developers without privileges who do not later receive privileges. We aim at analyzing whether they trigger different events with their actions, work on more issues, or do anything else that makes their behavior different from the behavior of developers who are not later granted privileges.

RQ1: How does the behavior of developers with privileges before they receive them differ from the behavior of developers without privileges?

3.1.2 *RQ2: Behavior Change After Privileges*

After receiving privileges, developers can perform different actions than before. We aim at investigating whether they use their privileges frequently, and if so, whether this results in a decrease of tasks that do not require privileges, or if they maintain the same workload but add additional tasks that require privileges.

RQ2: Does the behavior of developers change after receiving privileges relative to the behavior before receiving privileges?

3.1.3 *RQ3: Centrality Change After Privileges*

Furthermore, we are interested in whether developers become more central in the project after being granted privileges. We aim at analyzing whether they interact more with other (also central) developers in the project than before receiving privileges and whether they collaborate more with other (also central) developers on different files.

RQ3: Does the centrality of developers in the communication network or in the collaboration network change after receiving privileges compared to before?

3.2 EXPECTATIONS

Hypothesis 1: Developers with privileges are much more engaged in the project than developers without privileges, also before they receive privileges. They trigger more events than developers who do not receive privileges later, and in particular more relevant events (e.g., `reviewed`, `state_updated`, `commit_added`) instead of rather unimportant events (e.g., `renamed`, `commented`).

Explanation: If a developer is given privileges, it is most likely because they have done a good job before. We therefore expect them to stand out positively from all developers and trigger the "most important" or most influential events of all events for which no privileges are required. The hypothesis is validated for each event uniquely.

Hypothesis 2a: After receiving privileges, developers increasingly trigger privileged events, i.e., consistently and over a longer period of time.

Explanation: This hypothesis is based on the assumption that a developer is given privileges as a reward for their performance and trustworthiness. Such developers are also implicitly expected to fulfill the tasks entrusted to them and to actively participate in the coordination and guidance of project development, which in turn leads to privileged events being triggered on a regular basis.

Hypothesis 2b: After receiving privileges, the overall number of triggered issue events (common, extended and privileged events) increases.

Explanation: As explained in Hypothesis 2a, we assume that developers are given privileges as a reward for good performance. We expect these developers to maintain, and most likely increase, their level of activity because they are now even more motivated and competent. As a result, we expect them to contribute more to issue discussions and reviews, for example.

Hypothesis 3a: Developers have an increased centrality with regard to communication networks (i.e., issue networks) in the project after being granted privileges.

Explanation: The hypothesis of the communication centrality being increased after gaining privileges is based on the greater responsibility and leadership role that comes with privileges. These developers are most likely to perform more maintenance activities and are likely to communicate with other highly involved developers, all of which leads to increased

centrality regarding communication networks.

Hypothesis 3b: Developers do not have an increased centrality in the project in terms of collaboration networks (i.e., cochange networks) after being granted privileges.

Explanation: The hypothesis that collaboration centrality does not increase after gaining privileges is based on the explanation of hypothesis 3a. Developers who are busy with maintenance work do not have as much time to work on code. Thus, based on Hypothesis 3a, we expect them to shift their focus to maintenance and not spend more time working on code.

3.3 PREPARATION OF DATA

To investigate possible changes in the behavior of privileged developers and to understand the differences between privileged and unprivileged developers, we use GitHub issues and commits, which are explained in Section 2.1.

As a first step, we need to extract the issue and commit data from GitHub. Using the tool Codeface[1] [28, 29, 31], we extract commit metadata from Git, including the developer's name, e-mail address, date, and names of the modified files in a commit [4]. The issue metadata is extracted from GitHub using the tool GitHubWrapper[2]. This data includes which user initiated which event within which issue or pull-request, along with the corresponding timestamps [4]. We then use the tool Codeface Extraction[3] to merge the commit and issue data to match developers based on the same name or e-mail address [4].

It is important to notice that the usage of bots in GitHub projects is growing. These bots can perform tasks such as reviewing pull requests or closing abandoned issues [9, 22, 47–49]. Without filtering out these activities from the issue data, each bot would be treated as a developer, which would distort our analysis of the behavior of "real" developers. For this reason, we remove all events triggered by bots from the extracted issue data using the same procedure as Bock et al. [4].

From the extracted issue and commit data, we can obtain different relevant information like the date of the first privileged event from each developer, all triggered events, or the number of commits and issue events.

However, developer contributions can vary widely over the course of a project. Some developers are active for the entire project duration, while others are active for only a few months. Therefore, it is necessary to divide the data into standardized sections to allow for a more accurate comparison of the values of different developers. Bock et al. [4] investigated 25 different projects and discovered that in "all but one project more than 88 % of the developers using privileged events have a median time difference of less than 6 months between these events" [4]. Since we use a subset of the 25 investigated projects, this assumption also applies to our research. Given that privileged events occur at regular

1 https://github.com/se-sic/codeface/ (accessed 2024-03-21)
2 https://github.com/se-sic/GitHubWrapper/ (accessed 2024-03-21)
3 https://github.com/se-sic/codeface-extraction (accessed 2024-03-12)

intervals of less than 6 months, we divide all events into time windows of 6 months to identify trends and patterns, which is also common in the literature [25, 39].

From each split, we construct communication and collaboration networks (see Section 2.4) using the R library CORONET[4]. The constructed networks are not directed (i.e., the temporal order of interactions is ignored) and not simplified (i.e., multiple edges between a pair of developers are allowed, loops are allowed). From these networks, we later obtain centrality information.

If only consecutive windows were used, two developers who have worked on the same data source (commits or issues) within 6 months might not be connected by an edge because the events were separated by splitting. Missing connections between developers could affect eigenvector centrality, especially if edges between two central developers are dropped. Consequently, this scenario could have a negative impact on network metrics. Therefore, we split our data into sliding windows, where subsequent windows are shifted by half of the time window, i.e., 3 months, "such that subsequent windows overlap and cover the edges which are neglected when not using such a sliding-window approach" [4].

Before starting the actual analysis, we do one last preparatory step, which is to assign each developer to the group of privileged or unprivileged developers. To do this, we go through the list of triggered events of each developer and check if this developer has triggered a privileged event. Since we are using sliding windows, the first privileged event is included in two time windows. Any developer who has triggered a privileged event at least once is considered a privileged developer from that point on for the rest of their active time in the project, which is discussed further in Section 4.4.1. All other developers who have never triggered a privileged event belong to the group of unprivileged developers.

3.4 ANALYSIS AND IMPLEMENTATION

Now that we have the commit and issue data, the networks, and the assignment of developers to privileged and unprivileged, we can start performing the analyses to find answers to our research questions.

3.4.1 *RQ1: Privileged vs. Unprivileged Behavior*

To answer RQ1, we evaluate and compare the behavior of each group of developers, the privileged and the unprivileged.

For the developers without privileges, we consider all available data, that is, all commits and issues from the entire project period. For the group of developers with privileges, on the other hand, we only use data from the time before the privileges were granted, as we are interested in whether their behavior before receiving the privileges differs from the behavior of unprivileged developers. For the group of privileged developers, we additionally drop all data from the time windows in which the privileges were granted. The reason behind this decision is that we cannot use the entire time window, as using data from the period

4 https://github.com/se-sic/coronet (accessed 2024-03-12)

after receiving privileges would distort the results. The alternative would be to use the time windows and analyze only the data up to the date of the privilege granting. However, this is not very reliable since we later calculate the mean value over all time windows per developer. As we are not using the entire time window in which privileges are granted, this would distort the results because the values in this time window would be from a shorter time period than the values in all other time windows.

However, there may be outliers, such as developers whose first activity occurred in the first 3 months of the project. These developers may be project owners who could potentially distort the results by pushing the original project to the repository when it is opened. This results in a massive number of commits and activities that might not be comparable to the activities of other developers. To avoid this bias, we perform our analysis twice. First, we compare all privileged developers with all unprivileged developers, and then we compare all but the outliers, i.e., we exclude all developers (privileged and unprivileged) whose first activity occurred within the first 3 months of the project. We refer to these developers as "initial members" from now on.

The first step of our analysis is to achieve the number of issue events and commits from each developer in both groups. To accomplish this, we sum up the total number of triggered issue events and the total number of commits per developer over all considered time windows. In addition, we calculate the mean value of both values for each time window per developer to enable a more accurate comparison, as the time span in which developers work on a project can vary greatly.

The second step is to count how often the exact events (e.g. `commented`, `commit_added`, `mentioned`) were triggered by the two groups. To achieve this, we iterate through the list of issue events per developer in each time window and sum up the occurrences of each event. Then, we calculate the means of the occurrences of each event per developer per time window.

We perform both steps (i.e. number of commits, total number of issue events, and exact issue events) once for the entire group of privileged developers and the entire group of non-privileged developers, and once without initial members (i.e., developers whose first activity took place in the first 3months of the project) in both groups.

The resulting values of the performed steps (i.e., the total number of commits and issue events, the means of commits, issue events, and exact issue events) are then compared using the Mann-Whitney U test, as this test is well suited for non-normally distributed data and provides us with a comprehensive understanding of the differences between the groups. The Mann-Whitney U test is performed twice, once with the values from all developers and once with the values where initial members are excluded.
We use the null hypothesis that there is no significant difference between the distributions of the values of privileged and unprivileged developers. The alternative hypothesis is that the distribution of the values of the privileged developers are greater than those of the unprivileged developers.
The result of a Mann-Whitney U test then is a p-value, indicating the probability of obtaining the observed data if the assumption of no difference between the groups (null hypothesis) were true.

The smaller the p-value obtained from the test, the more significant the observed difference between the two groups, with significance defined in our thesis as a p-value below 0.05. When $p < 0.05$, we can reject the null hypothesis in favor of the alternative hypothesis.

In addition to the Mann-Whitney U test, we use Cliff's Delta[5] to determine the effect size, i.e., the extent of the difference between the values of the two groups. The resulting delta (δ) varies between 1 and -1, where a positive delta indicates that the values in the first group are greater than those in the second, and a negative delta indicates the opposite. A delta close to 1 or -1 indicates significant dominance of one group over the other. If δ is close to 0, the groups are stochastically equal [36].

3.4.2 *RQ2: Behavior Change After Privileges*

To answer RQ2, we analyze the behavior of developers with privileges after receiving privileges and compare it to their behavior before receiving privileges. For this purpose, we again analyze the number of issue events, the number of commits, and the exact events that were triggered by each developer, but only for the group of developers with privileges.

For each developer, we compare all time windows before the two time windows that include the first privileged event and all time windows after those two time windows, to get a direct comparison of behavior. Figure 3.1 illustrates the time windows that include the privilege receipt when using sliding windows. It is important to note that we do not include the two time windows during which a developer received privileges, because those time windows do not correspond to the time windows before or after receiving privileges, and therefore are not valuable for our analysis. The reason why we do not use these time windows instead, and only use the part of the time window that belongs to before or after receiving, is the same as for Section 3.4.1.

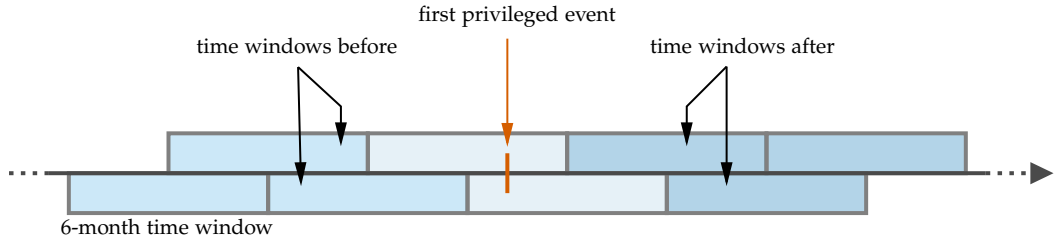

Figure 3.1: Time windows before and after the first privileged event when using sliding windows.

Comparing only one time window before and one time window after receiving privileges is not sufficient, as looking at only 6 months is not enough to get a proper understanding of a developer's behavior. During that particular time window, a developer may have been unavailable due to illness or vacation, or they may have been highly motivated immediately after receiving privileges, but experienced a significant drop in engagement after, for example, 6 months of receiving privileges.
However, comparing all time windows, including the first time window of a developer, is also not a good option because developers may behave differently in the initial phase of a

5 https://search.r-project.org/CRAN/refmans/rcompanion/html/cliffDelta.html (accessed 2024-03-15)

project than in their usual workflow. For instance, they need to familiarize themselves with the project and gain an overview [16, 34]. Therefore, we also drop the two time windows during which the first activity of a developer occurred.

Here again, we perform our analysis twice, once with all privileged developers and once with all but the initial members, which are, as described in Section 3.4.1, those whose first activity occurred in the first 3 months of the project.

The first step in the analysis is again to sum, for each privileged developer, the total number of issue events and commits that occurred in the time windows considered before the first privileged event (i.e., all but the two very first time windows) and after the first privileged event. In addition, we again calculate the mean of the counted issue events and commits per time window for each developer, once before and once after receiving privileges. With the mean value, we can compare their contributions more precisely, as developers may have been privileged for longer than unprivileged, or vice versa.

We also again count the exact events that were triggered in the time windows before and after receiving privileges by summing up the occurrences of each single event. Afterward, we calculate how often each event was triggered on average by each developer per time window before and respectively after the granting of privileges.

Both steps are performed once for all privileged developers and once for all except the initial members, i.e., those whose first activity occurred in the first 3 months of the project.

We now have two values for each measurement (the total number of commits and events for each developer, and the average number of commits, events, and exact events over all time windows), one from before and one from after the privileges were granted. This allows us to compare the resulting paired data using the Wilcoxon signed-rank test. The test is performed twice, once with the values from all privileged developers, and once with the values where initial members have been excluded.

The Wilcoxon signed-rank test is a suitable choice since it does not require a normal distribution and is robust to outliers. It reveals potentially significant changes in the measured values and, therefore, indicates if and in which way the behavior of the developers has changed. We use the null hypothesis that there is no significant difference between the values of the first group (i.e., before receiving privileges) and the values of the second group (i.e., after receiving privileges), and the alternative hypothesis that there is a tendency for the values of the first group to be smaller than the corresponding values of the second group.
The test then provides a p-value, which serves as an indicator of the extent to which the data contradicts the null hypothesis. If p falls below 0.05, it suggests significant evidence against the null hypothesis, leading to its rejection in favor of the alternative hypothesis.

Along with the Wilcoxon signed-rank test, we also conduct an analysis to derive the effect size R[6], which measures the significance of the difference between the paired samples. The resulting effect size R ranges from 1 to -1. It is positive when the data in the first group are greater than in the second group, and negative when the data in the second group

6 https://www.rdocumentation.org/packages/rcompanion/versions/2.3.7/topics/wilcoxonPairedR

are greater than in the first group [37]. This effect size makes it easier to understand the practical significance of the change or discrepancy found between the two groups.

3.4.3 *RQ3: Centrality Change After Privileges*

To answer RQ3, we use the eigenvector centrality (explained in Section 2.4) of each developer with privileges. We conduct two analyses, one based on an issue network and another based on a cochange network, to obtain centrality values for both networks, since the centrality values of a developer may differ between the two networks because edges between developers are created differently (see Section 2.4).

Similar to Section 3.4.2, we exclude time windows during which a developer received privileges, as they were both privileged and unprivileged during these time windows, providing no further information for RQ3. We again drop the two time windows in which the first activity of a developer occurred. For the analysis, we therefore consider all time windows except the initial ones before receiving privileges, and all time windows after receiving privileges for each developer.

We compute the eigenvector centrality for each developer in each considered time window, once from the issue network and once from the cochange network. Subsequently, for each developer, we calculate the mean of the values obtained for the two centralities (i.e., issue and cochange centrality) over all considered time windows before and after receiving privileges, respectively.

Here, we again have paired values since we have a cochange centrality from before receiving privileges and a cochange centrality from after receiving privileges for each developer, as well as for the issue centrality. Once more, we compare these paired values using the Wilcoxon signed-rank test, since our data are not necessarily normally distributed.
For both centralities, we use the null hypothesis that there are no differences between the centralities from before and after receiving privileges. For issue centrality, we use the alternative hypothesis that there is a tendency for the values from before receiving privileges to be smaller than the corresponding values from after receiving privileges. For cochange centrality, on the other hand, we use the alternative hypothesis that there is a difference between both groups, without stating whether the difference is positive or negative.
The test then provides a p-value, which serves as an indicator of the extent to which the data contradicts the null hypothesis. If p falls below 0.05, it suggests significant evidence against the null hypothesis, leading to its rejection in favor of the alternative hypothesis. Additionally, we perform an analysis to calculate the effect size R, as described in Section 3.4.2.

4

EVALUATION OF THE PROPOSED HYPOTHESES

In this chapter, we present the results of our analyses, starting with an overview of the projects used in our research. Subsequently, we evaluate the hypotheses by presenting and discussing the obtained results. Finally, we discuss the potential threats to the validity of our research.

4.1 PROJECTS

We chose 9 different OSS projects hosted on GitHub for our research. The development of these projects has been active for several years, and we hold data from a range of 2.5 to 6.5 years for each project.

We selected the following projects:

- Angular is a TypeScript-based web application framework.
- Atom is a text editor based on Electron.
- Bootstrap is a front-end framework providing a collection of CSS and JavaScript components for building websites and web applications.
- Deno is a runtime for executing JavaScript and TypeScript outside of the web browser.
- Electron is a framework that allows to build cross-platform desktop applications.
- Keras is a deep learning framework to efficiently build and train neural networks.
- OpenSSL is a library providing cryptographic functionalities for Transport Layer Security.
- VSCode is a code editor with support for various programming languages and extensions.
- Vue is a progressive JavaScript framework to build user interfaces.

The selected projects provide extensive coverage of OSS projects in general, with the number of developers ranging from 3172 to 68 664, the number of commits ranging from 2257 to 49 817, and the number of issues ranging from 8760 to 111 072.

Specific numbers for each project, including the number of commits, the number of issues, and the time range of available data, are detailed in Table 4.1.

Table 4.2 provides information about the number of developers in each project, including privileged developers, developers involved in issues, and developers contributing commits.

Table 4.1: Overview of commit and issue activity, along with the observation period, for the selected projects.

Project	#Commits	#Issues	Observation Period
Angular	12 357	38 501	2014-9-18 - 2020-9-25
Atom	15 629	21 138	2012-1-21 - 2020-12-10
Bootstrap	2266	31 735	2011-8-19 - 2020-12-22
Deno	3431	8760	2018-5-29 - 2020-12-22
Electron	10 610	26 606	2013-5-6 - 2020-12-22
Keras	3473	13 468	2015-3-28 - 2019-11-6
OpenSSL	8914	11 072	2013-5-13 - 2020-2-17
VSCode	49 817	111 072	2015-11-13 - 2020-12-22
Vue	2257	9325	2016-4-11 - 2020-11-24

Table 4.2: Summary of developer statistics including the total number of developers (Devs), privileged developers, privileged developers without initial members (w/o) and developers involved in issues and commits in the selected projects.

Project	#Devs	#Privileged Devs	#Privileged Devs w/o	#Devs in Issues	#Devs in Commits
Angular	23 420	70	51	667	22 856
Atom	21 388	54	51	298	21 045
Bootstrap	25 252	21	19	219	24 737
Deno	3172	9	3	3070	348
Electron	15 837	43	39	15 554	391
Keras	13 602	43	41	12 687	716
OpenSSL	8073	19	16	3419	408
VSCode	68 664	52	31	1001	67 877
Vue	9855	14	6	8751	217

4.2 RESULTS

Before presenting the detailed results for each research question, we first present some general findings to provide an overview of the projects and developer behavior.
In total, we analyze 9 OSS projects with varying amounts of privileged developers, ranging from 9 to 70. These developers make up between 0.076 % and 0.32 % of all developers, and it takes between 10 days and 6.5 years for developers to receive privileges. Considering the average number of occurrences of each issue event across all projects, the most frequently triggered event is `commented`. The most frequently triggered privileged event is `merged`, followed by `added_to_project` and `moved_columns_in_project`.

4.2.1 *RQ1: Privileged vs. Unprivileged Behavior*

For the following evaluation, the first group is Group_{FP} and refers to the group of privileged developers before they receive privileges (i.e., future privileged developers), and the second group is Group_{UP} and refers to developers who do not receive privileges later (i.e., unprivileged developers). A positive effect size therefore indicates that the values of future privileged developers are significantly greater than those of unprivileged developers, while a negative effect size indicates the opposite. We define significance as $p < 0.05$. Tables providing the exact results for each comparison can be found in the Appendix A.

For the total and average number of commits and issue events per developer, we observe significant results ($p < 2.054 \times 10^{-4}$) with an effect size of $0.596 < \delta < 0.986$. The comparison of Group_{FP} and Group_{UP} with respect to the median of the average number of commits and issue events is illustrated in Figure 4.1.

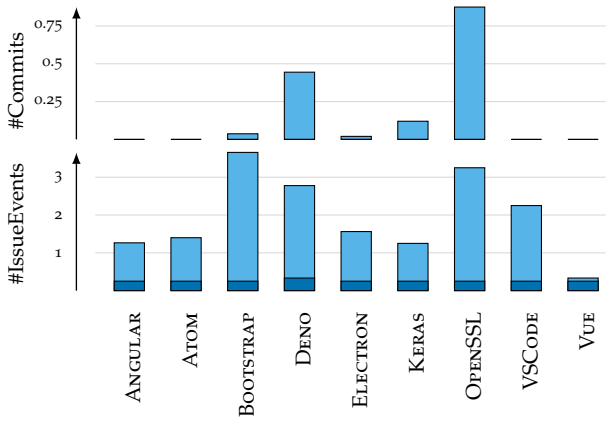

Figure 4.1: Comparison of Group_{FP} and Group_{UP}: Median of the average number of commits and issue events per developer. Dark blue represents Group_{UP}, light blue represents Group_{FP}.

For events, we generally observe 3 different outcomes. This results in a clustering of events into 3 categories, which are listed in Table 4.3.

The results of the events in Category 1.1 show high significance ($p < 1.439 \times 10^{-4}$), but varying effect sizes for the different projects.
`review_requested` has effect sizes of $\delta < 0.487$. The effect sizes of `head_ref_deleted`,

Table 4.3: Categories of events for RQ1

Category 1.1 High significance	Category 1.2 Mixed significance	Category 1.3 Not occuring
add_link, commented, commit_added, created, head_ref_deleted, mentioned, referenced_by, renamed, review_requested, reviewed	assigned, base_ref_changed, base_ref_force_pushed, comment_deleted, convert_to_draft, demilestoned, head_ref_force_pushed, head_ref_restored, labeled, marked_as_duplicate, milestoned, ready_for_review, referenced, resolution_updated, review_request_removed, state_updated, subscribed, type_updated, unassigned, unlabeled, unsubscribed	automatic_base_change_failed, automatic_base_change_succeeded, connected, cross_referenced, disconnected, unmarked_as_duplicate

renamed, and reviewed vary between 0.146 and 0.988. Regarding add_link, commented, commit_added, created, mentioned, and referenced_by, the effect sizes vary between 0.394 and 0.981.

The events in Category 1.2 show mixed results, as the significance and effect size of each event varies from project to project. This category can be divided into two subcategories: The first subcategory is demilestoned, milestoned, and resolution_updated. These events do not occur in every project. However, in all projects where they do occur, they show significant results. The latter event only occurs in Atom, Electron, VSCode and Vue. In these projects, it shows significant results ($p < 7.747 \times 10^{-138}$) with effect sizes of $0.133 < \delta < 0.182$. The other two events occur only in Angular, OpenSSL and VSCode. They show significant results ($p < 5.706 \times 10^{-193}$) with effect sizes of $0.086 < \delta < 0.5$.
The second subcategory is assigned, base_ref_changed, base_ref_force_pushed, comment_deleted, convert_to_draft, head_ref_force_pushed, head_ref_restored, labeled, marked_as_duplicate, ready_for_review, referenced, review_request_removed, state_updated, subscribed, type_updated, unassigned, unlabeled, and unsubscribed. This subcategory does not show significant results consistently across all projects. Each event shows significant results in only about half of the selected projects, details can be found in Table 4.4. In projects where significant results are observed, p is less than 0.03, and the effect sizes are positive and less than 0.859.

The events in Category 1.3 do not occur in any of the selected projects, and therefore cannot reveal differences between the two groups of developers.

We performed the analysis twice, once with the initial members included and once without, as described in Section 3.4.1. We found only minor differences between the two analyses, with only 4 events in total showing different results. Furthermore, the results of these events differed between the two analyses in only one project. As these differences are negligible for our analysis, we do not distinguish between results with and without initial members.

Table 4.4: Events in the second subcategory of Category 1.2 (mixed significance) from RQ1 and their results regarding the selected projects.

Event	Angular	Atom	Bootstrap	Deno	Electron	Keras	OpenSSL	VSCode	Vue
assigned	✓	✓	✗	✗	✓	✓	✓	✓	✓
base_ref_changed	✓	✗	✗	✗	✓	✗	✗	✗	✗
base_ref_force_pushed	✗	✗	✗	✗	✓	✗	✗	✗	✗
comment_deleted	✗	✗	✗	✗	✗	✗	✓	✗	✗
convert_to_draft	✗	✗	✗	✓	✗	✗	✗	✗	✗
head_ref_force_pushed	✓	✓	✓	(✓)	✓	✓	✓	✓	✓
head_ref_restored	✓	✗	✓	✗	✓	✓	✓	✓	✗
labeled	✓	✓	✓	✗	✓	✗	✓	✓	✓
marked_as_duplicate	✓	✓	✗	✗	✓	✗	✓	✓	✓
ready_for_review	✓	✗	✓	✓	✓	✗	✗	✗	✗
referenced	✓	✓	✓	✗	✓	✓	✗	✓	✓
review_request_removed	✓	✗	✗	✗	✓	✗	✓	✓	✗
state_updated	✓	✓	✓	(✓)	✓	✓	✓	✓	✓
subscribed	✓	✓	✓	(✓)	✓	✓	✓	✓	✓
type_updated	(✓)	✓	✗	✗	✓	✗	✗	✓	✓
unassigned	✓	✓	✗	✗	✓	✓	✓	✓	✓
unlabeled	✓	✓	✗	✗	✓	✗	✓	✓	✓
unsubscribed	✓	✗	✗	✗	✗	✓	✓	✓	✗

✓ = significant result (i.e., $p < 0.05$)

(✓) = significant result when including initial members, insignificant otherwise

✗ = insignificant result

RQ1

Based on the results, we reject the null hypothesis for the total and average number of commits and issue events, as well as for all issue events in Category 1.1. Since the results for the events in Category 1.2 are inconclusive, we can neither accept nor reject the null hypothesis. Additionally, as the events in Category 1.3 do not occur in any of the projects, we are not able to make a decision regarding the null hypothesis.

4.2.2 *RQ2: Behavior Change After Privileges*

For the following evaluation, we define significance as $p < 0.05$. The first group in the analysis are privileged developers before receiving privileges, and the second group are the same developers after receiving privileges. Thus, a negative effect size indicates that the values of developers after receiving privileges are greater than before, while a positive effect size indicates that the values after receiving privileges are less than before. See the Appendix A for tables containing the exact results for each comparison.

Regarding the total and average number of commits and issue events, we could not find consistent results. For Deno, Keras, and Vue, we do not find significant results ($p > 0.074$), neither in terms of commits nor issue events.
Considering only the total number of commits, we find significant results ($p < 0.027$) for Atom, Angular, Bootstrap, Electron (only for the analysis that includes initial members), and VSCode, with effect sizes ranging from -0.820 to -0.348. However, looking at the mean number of commits, this leaves Angular (with initial members), Atom, and VSCode, with $p < 0.007$ and effect sizes between -0.4450 and -0.3420.
For the total number of issue events, we again find significant results ($p < 2.671 \times 10^{-5}$) for Atom, Angular, Bootstrap, Electron, and VSCode, with effect sizes between -0.837 and -0.599. For the mean number of issue events, we find significant results ($p < 0.038$) with effect sizes between -0.647 and -0.253 for Angular, OpenSSL, and VSCode (only when including initial members). Table 4.5 contains the significance of the results of comparing commit and issue events for all projects.

Table 4.5: Results of comparison of commits and issue events from RQ2

	Angular	Atom	Bootstrap	Deno	Electron	Keras	OpenSSL	VSCode	Vue
`Counted commits`	✓	✓	✓	✗	(✓)	✗	✗	✓	✗
`Mean of counted commits`	(✓)	✓	✗	✗	✗	✗	✗	✓	✗
`Counted events`	✓	✓	✓	✗	✓	✗	✗	✓	✗
`Mean of counted events`	✓	✗	✗	✗	✗	✗	✓	(✓)	✗

✓ = significant result (i.e., $p < 0.05$)
(✓) = significant result when including initial members, insignificant otherwise
✗ = insignificant result

In contrast to RQ1, we do not observe any event that shows significant results across all selected projects. The categorization of events is detailed in Table 4.6.

Table 4.6: Categories of events for RQ2

Category 2.1 Mixed significance	Category 2.2 No significance	Category 2.3 Not occurring
add_link, added_to_project, assigned, base_ref_changed, base_ref_force_pushed, commented, commit_added, connected, convert_to_draft, converted_note_to_issue, created, demilestoned, head_ref_deleted, head_ref_force_pushed, head_ref_restored, labeled, marked_as_duplicate, mentioned, merged, milestoned, moved_columns_in_project, pinned, ready_for_review, referenced, referenced_by, removed_from_project, renamed, resolution_updated, review_dismissed, review_request_removed, review_requested, reviewed, state_updated, subscribed, transferred, type_updated, unassigned, unlabeled, unpinned, unsubscribed	automatic_base_change_succeeded, comment_deleted, disconnected, locked, unlocked, unmarked_as_duplicate, user_blocked	automatic_base_change_failed, cross_referenced, deployed, deployment_environment_changed

The results of Category 2.1 show mixed significance and varying effect sizes, ranging per event from significant results ($p < 0.049$) with effect sizes between -0.956 and -0.367 to insignificant results, depending on the project. However, we also find events with p slightly above 0.05. For `assigned`, `convert_to_draft`, `labeled`, `resolution_updated`, `review_request_removed`, `unlabeled`, `merged`, and `pinned`, we find in one or at most two projects $p = 0.0502$ and effect sizes of $R = -0.915$. The exact breakdown of which events have significant or insignificant results for which projects, and which events do not occur at all in which projects, can be found in Table 4.7. This table also includes the differences that occurred between the regular analysis and the additional analysis where we exclude initial members, as these differences occurred only for events of Category 2.1.

The events in Category 2.2 do not show significant results in any project. Not every event is observed in every project, and if an event is absent in a particular project, it cannot not reveal significant differences. However, for every other event-project pair within Category 2.2 where the event is present in the specific project, we also do not observe significant results.

The events in Category 2.3 do not occur in any of the projects and therefore cannot have significant results.

RQ2

Since the results for the total and average number of commits and issues and for the events in Category 2.1 are inconclusive, we can neither reject nor accept the null hypothesis. For the events in Category 2.2, we accept the null hypothesis as we did not observe any significant results. The absence of events in Category 2.3 in all projects does not allow us to make a decision about the null hypothesis.

Table 4.7: Events in Category 2.1 (mixed significance) from RQ2 and their results regarding the selected projects.

Event	Angular	Atom	Bootstrap	Deno	Electron	Keras	OpenSSL	VSCode	Vue
Common & Extended events:									
`add_link`	✓	✓	(✓)	✗	✓	✗	✓	✓	✗
`assigned`	✓	✓	✓	m	✓	✗	✓	✓	✗
`base_ref_changed`	✓	✗	(✓)	✗	✓	o	✗	(✓)	✗
`base_ref_force_pushed`	(✓)	(✓)	(✓)	o	✗	✗	✗	o	✗
`commented`	✓	✓	(✓)	✗	✓	✗	✓	✓	✗
`commit_added`	✓	✓	✗	✗	✓	✗	✓	✓	✗
`connected`	✓	o	✗	o	o	o	o	(✓)	o
`convert_to_draft`	✓	✗	✓	✗	m	o	o	(✓)	o
`created`	✓	✓	✗	✗	✓	✗	✓	✓	✗
`demilestoned`	✓	o	✓	✗	o	o	✗	✓	✗
`head_ref_deleted`	✓	✓	✗	(✓)	✓	✗	✗	✓	✗
`head_ref_force_pushed`	✓	✓	✓	✗	✓	✗	✗	✓	✗
`head_ref_restored`	(✓)	✓	✗	✗	✓	✗	✓	(✓)	✗
`labeled`	✓	✓	✓	m	✓	m	✓	✓	✗
`marked_as_duplicate`	✓	✗	✓	✗	✓	o	✗	✓	✗
`mentioned`	✓	✓	(✓)	✗	✓	✗	✓	✓	✗
`milestoned`	✓	✗	✓	✗	o	o	✗	✓	✗
`ready_for_review`	✓	✓	✓	✗	✓	✗	✗	✓	o
`referenced`	✓	✓	✗	✗	(✓)	✗	✗	✓	✗
`referenced_by`	✓	✓	✓	✗	✓	✗	✓	✓	✗
`renamed`	✓	✓	(✓)	✗	✓	✗	✓	✓	✗
`resolution_updated`	✗	✓	m	✗	✗	✗	✗	✓	✗
`review_request_removed`	✓	✓	m	m	✗	✗	✓	✓	✗
`review_requested`	✓	✓	✓	✗	✓	✓	✓	✓	✗
`reviewed`	✓	✓	✓	✗	✓	✗	✓	✓	✗
`state_updated`	✓	✓	✓	(✓)	✓	✗	✓	✓	✗
`subscribed`	✓	✓	✓	✗	✓	✗	✓	✓	✗
`type_updated`	✗	✓	o	✗	✗	✗	✓	✓	✗
`unassigned`	✓	✓	✓	✗	✓	✗	✓	✓	✗
`unlabeled`	✓	✓	✓	✗	✓	m	✓	✓	✗
`unsubscribed`	✓	✓	(✓)	✗	✓	✗	✗	✓	✗
Privileged events:									
`added_to_project`	✓	o	✓	o	✓	o	✓	✗	✗
`converted_note_to_issue`	(✓)	o	✗	o	✗	✗	✗	o	o
`merged`	✓	✓	✓	m	✓	(✓)	✗	✓	✗
`moved_columns_in_project`	✓	o	✓	o	✓	✗	✓	✗	✗
`pinned`	(✓)	✗	✗	✗	m	o	✗	✓	o
`removed_from_project`	✓	o	✓	o	✓	o	✗	✗	✗
`review_dismissed`	✓	✗	✓	✗	✓	✓	✓	✓	✗
`transferred`	✓	✓	o	✗	✗	o	o	✓	✗
`unpinned`	✓	✗	✗	✗	✗	o	✗	✓	o

✓ = significant result (i.e., $p < 0.05$)
(✓) = significant result when including initial members, insignificant otherwise
m = p slightly above 0.05
✗ = insignificant result
o = event did not occur

4.2.3 *RQ3: Centrality Change After Privileges*

For the following evaluation, Group_{FP} is the first group and refers to privileged developers before they received privileges, while Group_{P} is the second group and refers to the same developers after they received privileges. A negative effect size therefore indicates that the centralities of developers after receiving privileges are greater than before, while a positive effect size indicates the opposite. We define significance as $p < 0.05$. Tables providing the exact results for each comparison can be found in the Appendix A.
Table 4.8 provides an overview of the results for all projects.

Table 4.8: Results of issue and cochange centrality of the selected projects

Centrality	Angular	Atom	Bootstrap	Deno	Keras	Electron	OpenSSL	VSCode	Vue
Issue	✓	✓	✓	✗	✗	✓	✗	✓	✗
Cochange	✓	✓	✗	✗	✗	✗	✗	✓	✗

✓ = significant result (i.e., $p < 0.05$)
✗ = insignificant result

Regarding the eigenvector centrality in the issue networks, we find significant results ($p < 0.026$) in Angular, Atom, Bootstrap, Electron and VSCode, paired with effect sizes varying between -0.720 and -0.302. In the remaining 4 projects, we find insignificant results ($p > 0.124$) paired with effect sizes between -0.594 and -0.212.

For the eigenvector centrality in the cochange networks, we find significant results ($p < 2.637 \times 10^{-6}$) in Angular, Atom and VSCode, all paired with effect sizes smaller than -0.626. The remaining 6 projects show insignificant results ($p > 0.109$), all paired with effect sizes varying between -0.446 and -0.054.

A visual comparison of the average eigenvector centrality of developers before and after privilege granting in issue and cochange networks of all projects is shown in Figure 4.2.

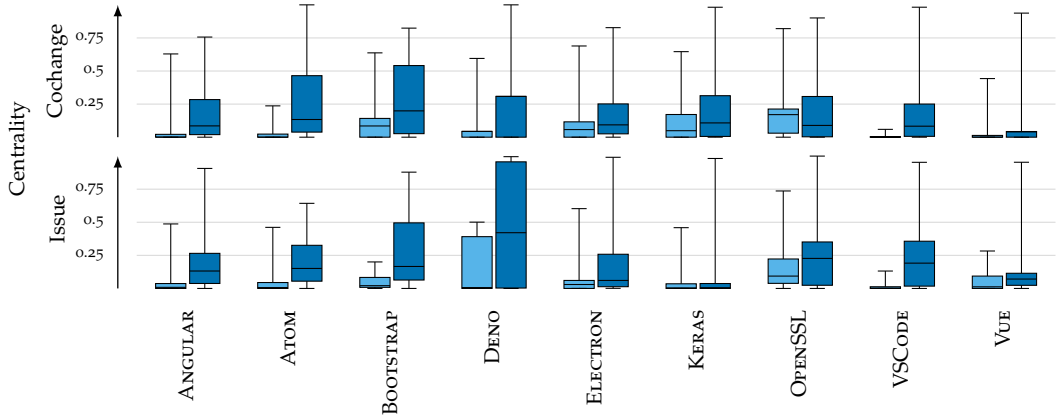

Figure 4.2: Eigenvector centralities of developers before receiving privileges (light blue) and developers after receiving privileges (dark blue), based on issue and cochange networks.

RQ3

Based on the results, we can reject the null hypothesis regarding issue networks for ANGULAR, ATOM, BOOTSTRAP, ELECTRON and VSCODE, and accept it for DENO, KERAS, OPENSSL and VUE.
The null hypothesis regarding cochange networks is rejected for ANGULAR, ATOM and VSCODE, and accepted for BOOTSTRAP, DENO, ELECTRON, KERAS, OPENSSL and VUE.

4.3 DISCUSSION

We now discuss the presented results to gain insights to answer our research questions.

4.3.1 *RQ1: Privileged vs. Unprivileged Behavior*

We are interested in whether the behavior of future privileged developers (Group_{FP}) differs from the behavior of unprivileged developers (Group_{UP}).

The total and average number of commits and triggered issue events of Group_{FP} is significantly higher than that of Group_{UP}, suggesting that Group_{FP} is putting more effort into the project. This trend of higher engagement is also apparent when considering the issue events separately. Group_{FP} triggers almost every issue event significantly more often than Group_{UP}.
Given this diverse involvement in significantly more issues and their contributions to significantly more commits compared to Group_{UP}, we can assume that both the number of commits and the number of issue events can serve as an indicator of a developer's qualification to receive privileges.

We also expect Group_{FP} to trigger relevant events such as `commit_added`, `reviewed`, and `state_updated` significantly more often compared to Group_{UP}, while we expect less significant differences for rather irrelevant events such as `renamed` and `commented` (see Section 3.2). The results of our analysis show that Group_{FP} indeed triggered `commit_added`, `reviewed` and `state_updated` significantly more often than Group_{UP}. However, `renamed` and `commented` were also triggered more often by Group_{FP} than by Group_{UP}. This suggests that, contrary to our expectations, these events are not irrelevant to future privileged developers. Since $\text{Group}_{\text{FP}}$is significantly more active in the project, developers in this group are likely to have a deeper understanding of and commitment to the project than developers who contribute only occasionally. As a result, Group_{FP} may be able to help when Group_{UP} has problems, and Group_{FP} may point out things that can be improved in code related to an issue. This in turn leads to Group_{FP} commenting on significantly more issues than Group_{UP}.

We also observe that some events reveal larger differences between Group_{FP} and Group_{UP} than others, indicated by a higher effect size δ. `commented`, `mentioned`, and `subscribed` reveal the biggest differences of all events. As mentioned above, we assume that `commented` is triggered much more often by Group_{FP} because of their broad knowledge of the project, which allows them to comment on a variety of different issues. This is in turn related to a high occurrence of `mentioned`, as there is a tight coupling between the two events when

having conversations with other developers. The high occurrence of `subscribed` can be explained by GitHub automatically subscribing developers to issues when certain actions occur[1], such as being assigned to the issue, commenting on it or being mentioned within it.

In summary, our analysis reveals that future privileged developers are indeed significantly more active than unprivileged developers, as evidenced by a significantly higher number of commits and triggered issue events per developer. The largest difference between future privileged and unprivileged developers appears in the occurrence of `commented`, `mentioned`, and `subscribed`, indicating that future privileged developers communicate more extensively with other developers.

4.3.2 *RQ2: Behavior Change After Privileges*

We are interested in whether the behavior of developers changes once they receive privileges. Specifically, we are interested in whether they trigger more events after receiving privileges, and whether they use their privileges at all.

The first thing to note is that the Vue project is an outlier in this research question. None of the privileged developers are truly active after receiving privileges, 92 % of them have an average of less than 1 commit per time window, and 85 % of them trigger an average of less than 7 issue events per time window. Therefore, the difference in behavior before and after receiving privileges is negligible. Hence, we do not include Vue in the following discussions.

For the events in Category 2.1 (see Table 4.6), we do not observe a consistent pattern over all projects. However, we identify an interesting pattern regarding the relation of project size and the number of significant results. Projects with a higher number of privileged developers, like Angular and VSCode (see Table 4.2), reveal lower numbers of events showing insignificant differences. In projects with a lower number of privileged developers, like Deno and Vue, no event reveals significant differences between developer contributions before and after receiving privileges.
However, there are deviations of this pattern. Even though Keras has a similar number of privileged developers to Electron, only 3 events (`review_requested`, `review_dismissed` and `merged`) show significant differences for Keras, while for Electron, 28 events reveal significant differences. This discrepancy might be due to a smaller number of commits and issues in Keras. We therefore assume that not only the number of privileged developers, but also the number of commits and issues in the project have an impact on whether significant differences are observed. There might also be other factors that affect whether and how developer contributions increase after privileges are granted, such as personal reasons (e.g., motivation or time availability) or the dynamics within the developer community of the project.

The differences in the total number of commits and triggered issue events by developers before and after receiving privileges also show the above-mentioned correlation between significant results and project size. In the 5 largest projects (i.e., Angular, Atom, Bootstrap,

1 https://docs.github.com/en/account-and-profile/managing-subscriptions-and-notifications-on-github/setting-up-notifications/about-notifications#notifications-and-subscriptions

ELECTRON, and VSCODE), we observe a significantly higher number of issue events and commits per developer after receiving privileges than before. Again, not only the number of privileged developers is a factor for finding significant differences, but also the number of issues and commits in the project. In the remaining 4 projects, we cannot find significant differences consistently.

When looking only at the common and extended events in the 4 projects with the highest number of issues (i.e., ANGULAR, BOOTSTRAP, ELECTRON, and VSCODE, see Table 4.1), it is visible that most of the events are triggered more often after receiving privileges than before. For smaller projects on the other hand, like DENO and OPENSSL, we did not find significant differences for all of these events, only for some. This suggests that, in smaller projects, only some events are triggered more often after receiving privileges, while others are triggered the same amount or even less often.
Considering only the privileged events in the 4 projects with the highest number of issues, we can see that `merged` is the only event that shows significant differences between developer contributions before and after privileges in all of the 4 projects.
`added_to_project`, `moved_columns_in_project`, `removed_from_project`, `review_dismissed` and `transferred` show significant differences in 3 of these 4 projects.
`converted_note_to_issue`, `pinned`, and `unpinned` show significant differences in, at most, 2 of the 4 projects. This indicates that, at least in large projects, developers do indeed use their privileges significantly often after receiving them. In smaller projects, however, we are not able to observe this behavior.

The observed contrast between larger and smaller projects can be due to different factors. A possible explanation for the relation between project size and the consistently significant differences between developer behavior before and after receiving privileges is that the larger number of developers with privileges increases the sample size. Consequently, the influence of less active privileged developers decreases in a larger sample. In addition, larger projects are likely to have more structured workflows and a greater need for active management and coordination, resulting in privileged developers being involved in administrative tasks that require regular triggering of issue events. Conversely, smaller projects might have more informal processes and less fixed structures that reduce administrative requirements. In such contexts, privileged developers may not be involved in as many administrative tasks or discussions, leading to smaller changes in behavior after receiving privileges. Thus, the observed discrepancy is likely due to the different sample sizes as well as the different organizational dynamics and project needs inherent in projects of different sizes.

We also observe that privileged events generally account for a rather small proportion of all events triggered after privileges are granted. Depending on the project, the sum of privileged events expressed as a percentage of all triggered events ranges from 0 % to 6.6 %.

The evaluation of our expectation that developers consistently trigger privileged events depends on the definition of "consistently". When defining "consistently" as triggering an average of > 1 privileged event per time window, the percentage of developers meeting this expectation varies between 15.2 % (KERAS) and 77.8 % (BOOTSTRAP) depending on the project. If it is defined as > 3, the percentage varies between 12.1 % (KERAS) and 72.2 % (BOOTSTRAP). When defined as > 10, the percentage varies between 12.1 % (KERAS) and 54 %

(VSCode).
Figure 4.3 illustrates the total number of developers consistently using their privileges, depending on the definition of consistence (i.e., the average number of privileges that has to be triggered per time window to be considered as consistently using privileges).

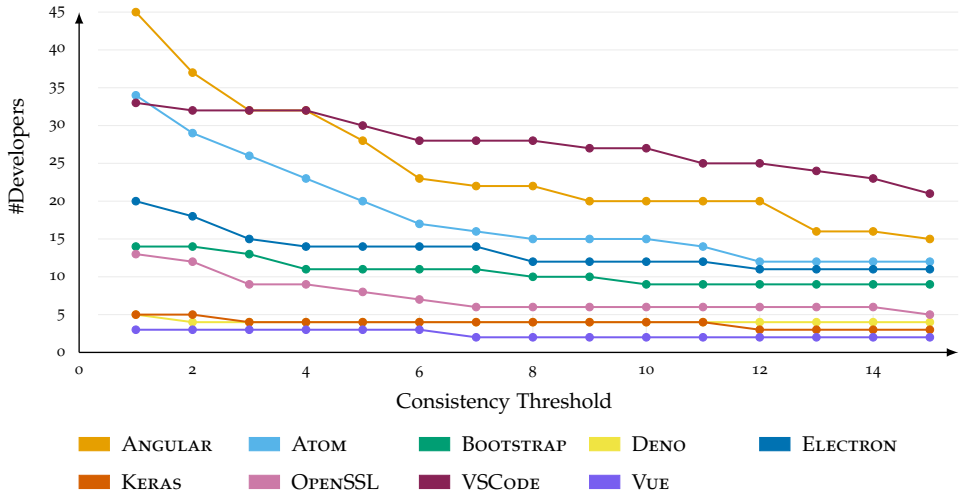

Figure 4.3: Number of developers consistently using their privileges, depending on the consistency threshold (i.e., the number of privileged events that must be triggered to be considered as consistently using privileges)

In summary, we observe that the change in developer behavior after receiving privileges varies from project to project, with project size in terms of the number of privileged developers, the number of issues, and the number of commits playing a key role in the extent of the behavioral change. In large projects, developers become more active after receiving privileges, as indicated by a significantly higher number of commits and triggered issue events. They also trigger significantly more common and extended events than before, however, of the privileged events, `merged` is the only one that is triggered significantly often.

4.3.3 *RQ3: Centrality Change After Privileges*

Finally, we are interested in whether the centralities of privileged developers in issue and cochange networks change after they receive privileges. We expect two different trends. In issue networks, we expect the eigenvector centrality to increase, but in cochange networks, we expect it to decrease, see Section 3.2.

We cannot find consistent results for all projects, but we again find a relationship between project size and significant differences, as in Section 4.3.2.

We start the evaluation with the eigenvector centrality in issue networks, which are constructed from GitHub issue data (explained in Section 2.4). Thus, in the following, we distinguish between projects with a high number of issues and those with a lower number based on the median (i.e., Atom).
Considering only the 5 projects with the highest number of issues (i.e., Angular, Atom, Bootstrap, Electron, VSCode, see Table 4.1), the eigenvector centrality of developers

in the issue networks increases significantly after receiving privileges. This indicates that developers do indeed work on more issues, and therefore communicate more with other developers, after receiving privileges.
For the remaining 4 projects, which have a much smaller number of issues, we do not observe a significant increase in the eigenvector centrality of privileged developers in the issue networks. When a project has fewer issues, privileged developers might not have the opportunity to work on a larger number of issues than before they were privileged. This limits the number of possible edges in the issue network, as developers are only connected by an edge if they have contributed to the same issue [4]. This in turn limits the centrality.
Our findings suggest that the threshold for significantly increasing eigenvector centrality in issue networks is $\approx 21\,000$ issues. This is demonstrated by the fact that the median project in terms of the number of issues, with $\approx 21\,000$ issues, is the smallest project showing significant differences.

We now evaluate the changes in eigenvector centrality in cochange networks. Since cochange networks are constructed from commit data (explained in Section 2.4), we distinguish between projects with a high number of commits and projects with a lower number of commits. First, we consider the 3 projects with the highest number of commits (i.e., ANGULAR, ATOM, VSCODE, see Table 4.1). In these projects, we observe that the eigenvector centrality of developers in the cochange networks changes significantly after receiving privileges. Since our alternative hypothesis in this case does not specify whether the values from before receiving privileges are greater or less than those from after receiving privileges, $p < 0.05$ does not reveal any information about this. However, since R is close to -1, we can assume that the centrality increases significantly after receiving privileges. This suggests that, contrary to our expectations, developers also spend more time working on the code, as a higher centrality in cochange networks indicates that the developer is editing many different files.
The remaining 6 projects have a smaller number of commits. In these projects, the eigenvector centrality does not increase significantly after receiving privileges. Similar to centrality in issue networks, the reason for this might be the smaller number of commits in the project, since this limits the interaction possibilities regarding contributions to the same file, which in turn limits the centrality regarding cochange networks.
These findings suggest that the threshold for significantly increasing eigenvector centrality in cochange networks is $\approx 12\,000$ commits, as the smallest project showing significant differences has $\approx 12\,000$ commits.

In summary, we observe that changes in eigenvector centrality after receiving privileges are related to the number of commits and issues in a project. For projects with a high number of issues (i.e., $> 21\,000$), our research consistently finds that the centrality of developers in issue networks increases after they receive privileges. The same applies to the relationship between the number of commits (i.e., $> 12\,000$) and cochange centrality. However, there may be other factors that explain the inconsistent results for centrality in both issue and cochange networks, such as the internal structure of the project or the dynamics of the community, which vary from project to project.

4.4 THREATS TO VALIDITY

After discussing the results of our research, we now outline the threats to the validity of our research, categorized into internal and external threats.

4.4.1 *Internal Validity*

The correctness and completeness of the data we use relies on the completeness of the GitHub data extracted with Codeface Extraction and the performance of Coronet, since we use this tool to build our networks. As these tools are well established and have been used in many previous studies, we assume that they work as intended.

We identify developers with privileges by searching the list of events they have triggered for an event that requires privileges to be triggered. Some developers may never trigger such an event, even if they belong to the group of developers with privileges. With our approach, we can only find those developers that in fact make use of their privileges. Nevertheless, we assume that the set of identified developers with privileges is sufficient for our study. We consider the unidentified developers to be exceptional and irrelevant to the analysis. Since they do not use their privileges, they should not be used as a benchmark for finding new candidates to be granted privileges.

The method we use to assign developers to the group of unprivileged or privileged developers (see Section 3.3) implies that once a developer has triggered a privileged event, that developer is considered a privileged developer for the rest of their time in the project. This status of being a privileged developer is never changed to unprivileged in our analyses, which could be a problem for developers who never use their privileges again after a certain point in time. However, this is also useful information for our research, as it also gives us insights into how developer behavior changes after receiving privileges. It could be a problem if a developer's privileges are revoked, but we assume that privileges are revoked either because a developer becomes inactive or because they do something that the other privileged developers do not appreciate. In the first case, this would not bias the results, since we are only analyzing data from the time the developers were active, and in the latter case, we assume that it would not bias the results either, since the developer would most likely stop working on the project if that happened.

We use only quantitative measures (i.e., the number of commits and the number of triggered issue events overall and for each event separately) for our analyses, and not qualitative ones. This might not capture the full complexity of developer behavior. Two developers might have the same number of commits and issue events, but the importance and impact of their contributions might differ greatly, because not all issue events are of equal importance to the project. Similarly, not all commits have the same importance, since some commits might contain large feature implementations, while others might contain only minor issues. Nevertheless, our approach remains informative for understanding developer behavior as it provides valuable insights that can serve as a starting point for further analysis.

When using sliding windows, there are two time windows in which a developer receives privileges, as the windows overlap. We chose to ignore both windows for all of our analyses

because there is mixed data in these windows, some of which occurred before and some after the privileges were granted. The full explanation of why we decided not to use these time windows can be found in Section 3.3. The alternative would be to split all data based on the date the privileges were received, which would allow us to analyze all time windows, since there would be no mixing of before and after privileges within a time window. This would have to be done on a per-developer basis, forcing us to build new networks for each developer, which would significantly increase the computational complexity of the analysis.

There are some cases where a developer's first activity occurs within the time window of privilege granting, for example, because they know the project owner and are granted privileges immediately. These developers then only have data (i.e., commits, issue events, and centrality values) for the time after they received privileges, since we ignore the time windows during which privileges were granted for our analysis. Other developers may have triggered a privileged event and proceeded to work on the project, but then stopped working within the window of privilege receipt. Since we do not use data from the exact time windows in which privileges were granted, we do not have data (i.e., commits, issue events, and centrality values) from after the privileges were granted for these developers. One possible method would be to ignore these developers. However, we decided not to ignore them, but to interpret this inactivity by putting zeros into the means we calculated. In this way, we can use them for RQ2 and RQ3 and compare their (possibly non-existent) behavior and centrality before receiving the privileges to their (possibly non-existent) behavior and centrality after receiving the privileges. The reason for our decision is that this type of behavior, where developers either stop working directly after receiving privileges or are granted privileges within the time window of their first activity, can provide valuable insights into the behavior of privileged developers and how their behavior changes after receiving privileges. However, we also decided to run the analyses without these developers as a sanity check.
For RQ2, 20 issue events show different results in 1 project each, while 6 issue events show different results in 2 projects each. `base_ref_force_pushed` shows different results in 3 projects. All but 2 of these 27 issue events show significant results in the general analysis, and insignificant results in the sanity check.
In the sanity check of RQ3, the difference between the centralities in the cochange networks becomes insignificant for Atom, as well as the difference between the centralities in the issue networks for Electron.
Nevertheless, as explained above, we are interested in the behavior change of all developers, including those who receive privileges directly or who stop working immediately after receiving privileges.

We also decided to compare all but the first time window of a developer from before privileges to all time windows after privileges for RQ2 and RQ3. Our analysis focuses only on time windows during which a developer was active, inactive time windows are ignored, and the analysis ends with the last time window in which a developer was active. However, it could be that developers slowly change their working behavior towards the end of their work time, or that they start into the project without working much and become really engaged at some point. These scenarios might bias our results, so we decided to provide a sanity check in the form of analyzing only the last time window of each developer before

receiving privileges to the first time window after receiving privileges for RQ2 and RQ3. For this check, we use all privileged developers, as we do in the regular analysis, i.e., including those who only have values for before or after the privileges.
For RQ2, the results regarding the number of commits do not differ from the regular analysis. The result for the number of triggered issue events differs only in VUE, where we observe significant differences in our sanity check.
The results of `demilestoned` and `milestoned` are equivalent to the results of the regular analysis. For all other events (except for those that never occur), we observe some differences between the results of the sanity check and those from the regular analysis. 9 events show a different result in exactly 1 project each, where the result changes from significant (in the regular analysis) to insignificant (in the sanity check). 15 events showed different results in 2 projects each, with the significance changing from insignificant to significant 4 times. The remaining 14 events showed different results in 3 to 4 projects each, and the results again changed from insignificant to significant 4 times. When using the results of the sanity check, `connected`, `converted_note_to_issue`, `convert_to_draft`, `pinned`, `transferred`, and `unpinned` are now insignificant for all projects. However, these variations are not unexpected, as we are only comparing two time windows of 6 months each, and it is quite possible that developers do not contribute equally over the entire period, and therefore some events do not occur in every time window.
For RQ3, the sanity check finds different results in the eigenvector centrality in cochange networks for KERAS and ELECTRON. These projects show insignificant results in our regular analysis, and significant results in the sanity check. However, the effect size in the sanity check is $-0.6 < R < -0.4$, which indicates that the centrality values from after receiving privileges are greater than those from before, but the effect is not extremely large. For the resulting projects, we find no differences in eigenvector centrality between the sanity check and the regular analysis. For eigenvector centrality in issue networks, VUE has significant differences (i.e., $p < 0.05$) when performing the sanity check, with an effect size of $R \approx -0.669$. These differences were also insignificant in our regular analysis.

4.4.2 *External Validity*

The external validity of our project might be compromised since we only use data from GITHUB projects and not from, e.g., GITLAB[2] projects. However, this is not a threat, as these platforms are almost identical and therefore should not bias our results.

Furthermore, it might be possible that the choice of projects used does not allow generalization. We avoid this by using a variety of projects of different sizes, with different numbers of (privileged) developers, and with different application domains for our analyses, which allows us to suggest that our results are transferable to other projects.

2 https://about.gitlab.com

5 RELATED WORK

In this chapter, we provide an overview of previous work related to this topic.

5.1 ESSENTIAL CHARACTERISTICS OF OSS MAINTAINERS

Dias et al. [15] provided research on several characteristics that are essential for the career success of OSS maintainers. The results show that there are indeed some important attributes, such as domain experience, technical excellence, motivation, diligence, and responsibility. Maintainers should "know the application domain, be aware of the technologies of the project, and have the experience to implement a quality process to perform code review and follow project quality standards" [15].
This is consistent with our findings that privileged developers are highly engaged in the project and take on more responsibility, as evidenced by performing administrative tasks such as labeling issues, and by commenting on many issues, indicating their deep and extensive knowledge of the project.
Even if they identified important attributes that a maintainer should have to be a great maintainer, they did not provide a solution for finding possible additional maintainers. Knowing the required traits is obviously helpful, but we still need to find clear metrics for identifying suitable candidates.

5.2 HIERARCHY IN OSS PROJECTS

Joblin et al. [28] showed that core developers hold higher positions in the hierarchy and peripheral developers hold lower ones. This goes along with the fact that the hierarchical part of OSS projects is occupied by the most active developers of the project and only rarely with other developers, as shown by Joblin et al. [30]. They stated the hypothesis that developers "who move up in the hierarchy tend to take more coordination tasks" [30], and also found that there is a correlation between the time a developer is involved in the project and their hierarchical position. In addition, developers in the hierarchical part are performing most of the file edits of the project. This approach confirms our assumption that developer behavior is crucial for getting a higher position in the hierarchy, which results in more responsibilities.
The hypothesis that developers who move up in the hierarchy take on more coordination tasks can be interpreted as being involved in issues and generally communicating with many other developers. In our work, this would be indicated by a higher centrality in issue networks, since these are constructed from communication data and developers are connected when they have contributed to the same issue. Since the eigenvector centrality of developers in issue networks does indeed increase after receiving privileges, our results

are consistent with their hypothesis. Developers who receive privileges take on more coordination tasks, suggesting that moving up in the hierarchy is equivalent to receiving privileges.
However, they also do not provide any method for identifying potential developers who could be granted privileges.

5.3 DEVELOPER ROLES AND MIGRATION BETWEEN ROLES

As described in Section 2.2, Ye and Kishida [50] found that developers can have different roles, being *project leader, core member, active developer, peripheral developer, bug fixer, bug reporter, reader,* and *passive user*. They state that the roles of OSS developers might constantly change depending on the efforts of developers. Roles are not preassigned but evolve dynamically: New members migrate from passive users to readers, and as they learn more about the system, start fixing bugs, and add new functionality, they become peripheral developers. As their developed programs are made publicly available, the community recognizes them. Depending on the wish and efforts of developers, as the recognition gets higher, they are trusted with more challenging tasks and finally, they can enter the "highly selected "inner circle" of core members" [50].
Terceiro et al. [44] found that the processes of migration between roles vary between different OSS projects. But in general, the main aspect of achieving central roles is merit-based. Developers who provide continuous and valuable contributions can become leaders. They also found that core developers contribute code of different complexity than peripheral developers.
Both studies are consistent with our findings that future privileged developers are more active and contribute significantly more. However, neither Ye and Kishida [50] nor Terceiro et al. [44] stated exactly how to find new core developers. They both only stated that developers who perform more challenging tasks and who contribute valuable code might join the core team. This still does not allow us to identify behavioral differences between privileged and unprivileged developers to help find suitable candidates to receive privileges.

5.4 AUTOMATIC IDENTIFICATION OF CORE DEVELOPERS

The work of Bock et al. [4] addresses the automatic identification of core developers and maintainers. They used the permissions that GitHub grants to users to classify GitHub issue events into 3 categories: common, extended, and privileged events (see Section 2.3). Common events can be triggered by anyone, extended events require, at least, triage permission, and privileged events require, at least, write permission.
They also distinguished between core and peripheral developers, and assigned developers to these groups based on the events they triggered. Any developer who triggered a privileged event was identified as a core developer, since they assumed that an event that requires, at least, write permission, is either critical for the maintenance of the project or for decision making, both of which distinguish a core developer.
We apply their classification of developers into core and peripheral developers and their mapping of GitHub's user permissions to these developer roles, shown in Table 2.1. But

this method still does not solve our problem as it "cannot be used to identify potential candidates for future core developers before they get any privileges in the project" [4].

5.5 BEHAVIOR OF ELITE DEVELOPERS

The work of Wang et al. [45] focused on the behavior of elite developers. They defined elite developers as developers with administrative privileges, i.e., write permission. This definition is very similar to the definition of privileged developers proposed by Bock et al. [4], which we adopt for our research, as they define them by holding write, maintain, or admin permissions.
According to the findings of Wang et al. [45], the work of elite developers includes more than just committing source code. They are active in "moderating discussions of an unfixed issue, documenting changes, organizing the project, and communicating with other contributors" [45]. To provide better insight, they categorized the GitHub events into high-level categories: *communicative, organizational, supportive,* and *typical*.
The key findings from their analyses were that elite developers participate in a variety of activities, with technical contributions representing only a small fraction. In addition, as the project grows, elite developers focus their efforts on supportive and communicative activities, resulting in a decreasing number of commits. Even as their focus shifts, the majority of commits are still contributed by elite developers. Overall, elite developers are the authors of the majority of events, and the number of activities performed is much higher in all categories compared to non-elite developers.
We also observe that, at least in large projects, privileged (i.e., elite) developers increasingly comment on issues and communicate with other contributors. However, we find that the number of commits made by developers after they receive privileges (i.e., after they become elite) does not decrease in every project, only in smaller ones. In large projects, the number of commits per developer increases after they receive privileges. Since the number of issue events per developer in these projects however also increases after receiving privileges, the higher number of commits does not completely contradict their hypothesis. Developers might actually shift their focus to support and communication activities, which leads to a higher number of issue events. But at the same time, they can still contribute more commits than before receiving privileges, perhaps just not in the same proportion as the increase in the number of issue events.
Although the work of Wang et al. [45] found interesting and meaningful comparisons, they did not analyze whether the behavior of elite developers differs from before they became elite, nor how to identify new elite developers.

6 CONCLUDING REMARKS

In this final chapter, we summarize our work and conclude the findings. Afterward, we provide some ideas for future work.

6.1 CONCLUSION

We aim at finding measures to identify developers who have the potential to be granted privileges by comparing the behavior of privileged and unprivileged developers in OSS projects. To achieve this goal, we extracted the issue and commit data from 9 OSS projects hosted on GitHub. We divided these data into sliding time windows of 6 months to allow for more precise analyses by using equally large time periods and balancing out very local contribution behavior. From each time window of data, we constructed an issue network and a cochange network using the R library Coronet. We then examined the GitHub data and the networks to understand how developers with privileges behave before they receive privileges, how this differs from the behavior of developers without privileges, and how the behavior of developers changes after they receive privileges.

First, we analyzed how the behavior of privileged developers before receiving privileges differs from the behavior of unprivileged developers. As expected, future privileged developers are significantly more active than unprivileged developers. The former have a higher number of commits and overall triggered issue events. This observation also applies when considering each issue event individually, with almost every event being triggered more frequently. The highest difference between both groups of developers is visible in issue events that are correlated with communication. `commented`, `mentioned`, and `subscribed` reveal the largest differences between future privileged and unprivileged developers.

Second, we analyzed if and how the behavior of developers changes after they receive privileges. We did not find a consistent pattern of behavior change across all the selected projects. However, we did find that there is a relationship between the size of a project (i.e., #privileged developers, #commits, #issues) and significant changes in developer behavior. In larger projects, there is a significant increase in developer activity after they receive privileges. Overall, they trigger more common and extended events, and they also trigger privileged events, although not to the expected extent. Conversely, in smaller projects, there are no consistent significant differences in behavior before and after receiving privileges.

Finally, we analyzed whether the average eigenvector centrality of each developer in the issue and cochange networks changes after receiving privileges. To determine this, we obtained the eigenvector centrality metric from the constructed networks. We did not find consistent results across all projects, but we again found a relationship between the size of a project and significant changes in the eigenvector centrality of developers. However,

this relationship does not depend on the number of privileged developers, but only on the number of commits for the centrality in cochange networks, and on the number of issues for the centrality in issue networks. We found that, in large projects, the eigenvector centrality of developers in both types of networks increases significantly after receiving privileges. In smaller projects, however, we did not find any significant differences.

In summary, we observe that future privileged developers are significantly more active than developers who are never granted privileges. This increased activity is apparent in both the average and total number of commits and issue events, as well as in their more frequent triggering of individual issue events.
This allows us to suggest that developers with similar behavior, that is, developers with a significantly higher number of commits and a significantly higher number of triggered issue events, are suitable candidates for receiving privileges in OSS projects. These candidates are characterized in particular by the fact that they increasingly comment on issues, are mentioned by other developers, and subscribe to issues.
We also observe that, at least in large projects, both the activity and the eigenvector centrality of developers increase after they receive privileges. They not only make use of their privileges but also trigger significantly more common and extended events. Therefore, we conclude that our insights on the behavioral characteristics of privileged developers are valuable and can be utilized to identify possible candidates to receive privileges.

6.2 FUTURE WORK

While this work is a first step toward identifying suitable developers to be granted privileges in OSS projects, there are several ways to improve it.

Our analyses focused on quantitative measures such as the number of triggered issue events, commits, and specific events triggered. One possible improvement to ensure more accurate suggestions might be to use not only quantitative measures but also qualitative ones, to gain more insight into how the contributions of privileged developers before they received privileges differ from those of developers who do not receive privileges. It would also be useful to find a precise threshold to define when a developer's contributions are considered "significantly higher" compared to the contributions of other developers. While we have identified the need for higher contributions, the exact determination of the point at which a contribution is "significant" remains unclear.

Although we analyzed 9 OSS projects of different sizes and application domains, the sample set could be expanded to gain a more comprehensive understanding and more reliable insights into the differences between developers. This expansion could also validate the thresholds we established for centrality increases. Furthermore, it would be interesting to know why exactly the threshold is 12 000 commits for cochange centrality and 21 000 issues for issue centrality.

A further step might also involve the implementation of algorithms based on our approach or improved methods. These algorithms could then be integrated into GitHub to automate the granting of privileges and the identification of suitable candidates.

A

APPENDIX

This chapter presents the statistical test results for all research questions in tables.

A.1 RESULTS FOR RQ1

Table A.1 to Table A.9 present the results of the one-tailed Mann-Whitney U tests for RQ1, comparing future privileged developers to unprivileged developers with regard to each issue event and the number of commits and issue events. The Mann-Whitney U test evaluates for each variable whether there is a significant difference in the distributions of the two groups, focusing on whether the distribution of values in the first group (future privileged developers) tends to be higher than that of the second group (unprivileged developers). The U-statistic represents the test statistic, while the corresponding p-value indicates whether the alternative hypothesis is accepted ($p < 0.05$) or not. Additionally, Cliff's Delta provides the corresponding effect size.

Table A.1: RQ1 Angular

Variable	U	p-value	Cliff's Delta
Counted commits	894 681.0	0.0	0.707
Counted events	1 003 362.5	0.0	0.915
Mean of counted commits	893 801.0	0.0	0.705
Mean of counted events	977 539.5	0.0	0.865
assigned	785 735.0	0.0	0.499
demilestoned	569 541.0	0.0	0.087
labeled	853 953.5	0.0	0.629
marked_as_duplicate	546 819.0	0.0	0.043
milestoned	626 488.0	0.0	0.195
resolution_updated	524 078.0	1.0	0.0
type_updated	535 448.5	0.0	0.022
unassigned	659 076.0	0.0	0.258
unlabeled	762 951.0	0.0	0.456
unmarked_as_duplicate	524 078.0	1.0	0.0
add_link	925 247.0	0.0	0.765
commit_added	918 012.5	0.0	0.752
referenced	638 760.5	0.0	0.219
automatic_base_change_failed	524 078.0	1.0	0.0
automatic_base_change_succeeded	524 078.0	1.0	0.0
base_ref_changed	546 655.5	0.0	0.043
base_ref_force_pushed	524 055.0	0.518	0.0
comment_deleted	524 078.0	1.0	0.0
commented	993 365.5	0.0	0.895
committed	524 078.0	1.0	0.0
connected	524 078.0	1.0	0.0
convert_to_draft	523 917.0	0.547	0.0
created	942 062.0	0.0	0.798
cross_referenced	524 078.0	1.0	0.0
disconnected	524 078.0	1.0	0.0
head_ref_deleted	755 070.0	0.0	0.441
head_ref_force_pushed	902 259.0	0.0	0.722
head_ref_restored	580 496.5	0.0	0.108
mentioned	931 043.0	0.0	0.777
ready_for_review	569 084.5	0.0	0.086
referenced_by	934 827.0	0.0	0.784
renamed	767 935.5	0.0	0.465
review_request_removed	580 817.0	0.0	0.108
review_requested	692 325.5	0.0	0.321
reviewed	769 452.0	0.0	0.468
state_updated	845 265.5	0.0	0.613
subscribed	859 726.0	0.0	0.64
unsubscribed	543 985.5	0.0	0.038

Table A.2: RQ1 Atom

Variable	U	p-value	Cliff's Delta
Counted commits	553 166.5	0.0	0.597
Counted events	683 891.0	0.0	0.975
Mean of counted commits	553 097.5	0.0	0.597
Mean of counted events	667 024.5	0.0	0.926
assigned	409 222.0	0.0	0.182
demilestoned	346 351.5	1.0	0.0
labeled	461 400.5	0.0	0.332
marked_as_duplicate	356 847.0	0.0	0.03
milestoned	346 351.5	1.0	0.0
resolution_updated	409 237.0	0.0	0.182
type_updated	440 620.5	0.0	0.272
unassigned	367 263.0	0.0	0.06
unlabeled	440 646.5	0.0	0.272
unmarked_as_duplicate	346 351.5	1.0	0.0
add_link	525 192.0	0.0	0.516
commit_added	483 492.0	0.0	0.396
referenced	427 945.5	0.0	0.236
automatic_base_change_failed	346 351.5	1.0	0.0
automatic_base_change_succeeded	346 351.5	1.0	0.0
base_ref_changed	346 302.0	0.528	0.0
base_ref_force_pushed	346 351.5	1.0	0.0
comment_deleted	346 335.0	0.516	0.0
commented	593 693.0	0.0	0.714
committed	346 351.5	1.0	0.0
connected	346 351.5	1.0	0.0
convert_to_draft	346 302.0	0.528	0.0
created	552 850.0	0.0	0.596
cross_referenced	346 351.5	1.0	0.0
disconnected	346 351.5	1.0	0.0
head_ref_deleted	437 935.0	0.0	0.264
head_ref_force_pushed	418 646.5	0.0	0.209
head_ref_restored	346 153.5	0.555	−0.001
mentioned	580 598.5	0.0	0.676
ready_for_review	346 269.0	0.536	0.0
referenced_by	541 574.0	0.0	0.564
renamed	470 805.5	0.0	0.359
review_request_removed	346 351.5	1.0	0.0
review_requested	367 244.0	0.0	0.06
reviewed	397 187.0	0.0	0.147
state_updated	500 080.0	0.0	0.444
subscribed	563 098.0	0.0	0.626
unsubscribed	343 926.0	0.685	−0.007

Table A.3: RQ1 Bootstrap

Variable	U	p-value	Cliff's Delta
Counted commits	320 774.5	0.0	0.73
Counted events	368 361.0	0.0	0.987
Mean of counted commits	320 102.5	0.0	0.727
Mean of counted events	316 703.0	0.0	0.708
assigned	185 362.5	0.51	0.0
demilestoned	185 370.0	1.0	0.0
labeled	196 922.5	0.0	0.062
marked_as_duplicate	185 370.0	1.0	0.0
milestoned	185 370.0	1.0	0.0
resolution_updated	185 370.0	1.0	0.0
type_updated	185 370.0	1.0	0.0
unassigned	185 362.5	0.51	0.0
unlabeled	185 370.0	1.0	0.0
unmarked_as_duplicate	185 370.0	1.0	0.0
add_link	351 284.0	0.0	0.895
commit_added	361 000.0	0.0	0.947
referenced	218 893.5	0.0	0.181
automatic_base_change_failed	185 370.0	1.0	0.0
automatic_base_change_succeeded	185 370.0	1.0	0.0
base_ref_changed	185 257.5	0.538	−0.001
base_ref_force_pushed	185 370.0	1.0	0.0
comment_deleted	185 370.0	1.0	0.0
commented	341 550.5	0.0	0.843
committed	185 370.0	1.0	0.0
connected	185 370.0	1.0	0.0
convert_to_draft	185 325.0	0.524	0.0
created	348 815.5	0.0	0.882
cross_referenced	185 370.0	1.0	0.0
disconnected	185 370.0	1.0	0.0
head_ref_deleted	319 009.5	0.0	0.721
head_ref_force_pushed	233 831.0	0.0	0.261
head_ref_restored	234 386.5	0.0	0.264
mentioned	337 652.0	0.0	0.822
ready_for_review	209 975.5	0.0	0.133
referenced_by	338 246.0	0.0	0.825
renamed	278 483.0	0.0	0.502
review_request_removed	185 370.0	1.0	0.0
review_requested	209 946.0	0.0	0.133
reviewed	258 431.5	0.0	0.394
state_updated	315 388.5	0.0	0.701
subscribed	326 054.0	0.0	0.759
unsubscribed	181 912.5	0.703	−0.019

Table A.4: RQ1 Deno

Variable	U	p-value	Cliff's Delta
Counted commits	12 163.0	0.0	0.987
Counted events	12 162.5	0.0	0.987
Mean of counted commits	12 148.0	0.0	0.985
Mean of counted events	12 136.5	0.0	0.983
assigned	6120.0	1.0	0.0
demilestoned	6120.0	1.0	0.0
labeled	6120.0	1.0	0.0
marked_as_duplicate	6120.0	1.0	0.0
milestoned	6120.0	1.0	0.0
resolution_updated	6120.0	1.0	0.0
type_updated	6120.0	1.0	0.0
unassigned	6120.0	1.0	0.0
unlabeled	6120.0	1.0	0.0
unmarked_as_duplicate	6120.0	1.0	0.0
add_link	11 935.5	0.0	0.95
commit_added	12 019.5	0.0	0.964
referenced	6026.0	0.599	−0.015
automatic_base_change_failed	6120.0	1.0	0.0
automatic_base_change_succeeded	6120.0	1.0	0.0
base_ref_changed	6112.0	0.531	−0.001
base_ref_force_pushed	6120.0	1.0	0.0
comment_deleted	6120.0	1.0	0.0
commented	12 120.0	0.0	0.98
committed	6120.0	1.0	0.0
connected	6120.0	1.0	0.0
convert_to_draft	9099.0	0.0	0.487
created	12 112.0	0.0	0.979
cross_referenced	6120.0	1.0	0.0
disconnected	6120.0	1.0	0.0
head_ref_deleted	10 558.5	0.0	0.725
head_ref_force_pushed	8995.0	0.0	0.47
head_ref_restored	6098.0	0.549	−0.004
mentioned	11 271.0	0.0	0.842
ready_for_review	10 631.5	0.0	0.737
referenced_by	12 108.5	0.0	0.979
renamed	10 530.0	0.0	0.721
review_request_removed	6116.0	0.523	−0.001
review_requested	9101.0	0.0	0.487
reviewed	12 163.5	0.0	0.988
state_updated	10 405.5	0.0	0.7
subscribed	11 051.0	0.001	0.806
unsubscribed	6004.0	0.61	−0.019

Table A.5: RQ1 Electron

Variable	U	p-value	Cliff's Delta
Counted commits	393 305.5	0.0	0.69
Counted events	460 287.0	0.0	0.978
Mean of counted commits	393 002.0	0.0	0.689
Mean of counted events	444 140.5	0.0	0.909
assigned	309 974.5	0.0	0.332
demilestoned	232 665.0	1.0	0.0
labeled	363 982.0	0.0	0.564
marked_as_duplicate	240 361.5	0.0	0.033
milestoned	232 665.0	1.0	0.0
resolution_updated	263 659.0	0.0	0.133
type_updated	294 525.5	0.0	0.266
unassigned	271 244.0	0.0	0.166
unlabeled	309 989.0	0.0	0.332
unmarked_as_duplicate	232 665.0	1.0	0.0
add_link	412 169.0	0.0	0.772
commit_added	395 274.0	0.0	0.699
referenced	263 843.0	0.0	0.134
automatic_base_change_failed	232 665.0	1.0	0.0
automatic_base_change_succeeded	232 665.0	1.0	0.0
base_ref_changed	247 967.0	0.0	0.066
base_ref_force_pushed	248 176.0	0.0	0.067
comment_deleted	232 665.0	1.0	0.0
commented	417 039.5	0.0	0.792
committed	232 665.0	1.0	0.0
connected	232 665.0	1.0	0.0
convert_to_draft	232 575.0	0.543	0.0
created	394 232.5	0.0	0.694
cross_referenced	232 665.0	1.0	0.0
disconnected	232 665.0	1.0	0.0
head_ref_deleted	337 251.5	0.0	0.45
head_ref_force_pushed	323 886.0	0.0	0.392
head_ref_restored	240 223.0	0.0	0.032
mentioned	408 935.5	0.0	0.758
ready_for_review	240 084.0	0.0	0.032
referenced_by	417 791.5	0.0	0.796
renamed	343 947.5	0.0	0.478
review_request_removed	263 638.0	0.0	0.133
review_requested	314 111.5	0.0	0.35
reviewed	339 151.0	0.0	0.458
state_updated	403 256.5	0.0	0.733
subscribed	388 582.5	0.0	0.67
unsubscribed	231 855.0	0.627	−0.003

Table A.6: RQ1 KERAS

Variable	U	p-value	Cliff's Delta
Counted commits	261 974.0	0.0	0.802
Counted events	274 958.0	0.0	0.891
Mean of counted commits	261 424.5	0.0	0.798
Mean of counted events	271 745.5	0.0	0.869
assigned	151 645.0	0.0	0.043
demilestoned	145 394.5	1.0	0.0
labeled	145 325.5	0.542	0.0
marked_as_duplicate	145 371.5	0.525	0.0
milestoned	145 394.5	1.0	0.0
resolution_updated	145 394.5	1.0	0.0
type_updated	145 394.5	1.0	0.0
unassigned	151 646.0	0.0	0.043
unlabeled	145 337.0	0.538	0.0
unmarked_as_duplicate	145 394.5	1.0	0.0
add_link	224 559.5	0.0	0.544
commit_added	256 355.5	0.0	0.763
referenced	150 505.0	0.03	0.035
automatic_base_change_failed	145 394.5	1.0	0.0
automatic_base_change_succeeded	145 394.5	1.0	0.0
base_ref_changed	145 337.0	0.538	0.0
base_ref_force_pushed	145 394.5	1.0	0.0
comment_deleted	145 383.0	0.518	0.0
commented	243 417.0	0.0	0.674
committed	145 394.5	1.0	0.0
connected	145 394.5	1.0	0.0
convert_to_draft	145 394.5	1.0	0.0
created	249 733.0	0.0	0.718
cross_referenced	145 394.5	1.0	0.0
disconnected	145 394.5	1.0	0.0
head_ref_deleted	212 796.0	0.0	0.464
head_ref_force_pushed	175 150.5	0.0	0.205
head_ref_restored	164 159.5	0.0	0.129
mentioned	225 686.5	0.0	0.552
ready_for_review	145 360.0	0.53	0.0
referenced_by	223 184.5	0.0	0.535
renamed	205 932.0	0.0	0.416
review_request_removed	145 394.5	1.0	0.0
review_requested	151 681.0	0.0	0.043
reviewed	194 761.5	0.0	0.34
state_updated	211 185.5	0.0	0.452
subscribed	210 555.0	0.0	0.448
unsubscribed	151 584.0	0.0	0.043

Table A.7: RQ1 OpenSSL

Variable	U	p-value	Cliff's Delta
Counted commits	59 158.5	0.0	0.933
Counted events	60 858.0	0.0	0.989
Mean of counted commits	58 651.0	0.0	0.917
Mean of counted events	60 504.5	0.0	0.977
assigned	47 562.5	0.0	0.554
demilestoned	42 494.5	0.0	0.389
labeled	46 322.0	0.0	0.514
marked_as_duplicate	32 300.0	0.0	0.056
milestoned	45 881.5	0.0	0.499
resolution_updated	30 600.0	1.0	0.0
type_updated	30 945.5	0.406	0.011
unassigned	35 692.0	0.0	0.166
unlabeled	47 565.0	0.0	0.554
unmarked_as_duplicate	30 600.0	1.0	0.0
add_link	58 035.5	0.0	0.897
commit_added	58 467.5	0.0	0.911
referenced	31 580.0	0.301	0.032
automatic_base_change_failed	30 600.0	1.0	0.0
automatic_base_change_succeeded	30 600.0	1.0	0.0
base_ref_changed	30 546.0	0.571	−0.002
base_ref_force_pushed	30 600.0	1.0	0.0
comment_deleted	32 300.0	0.0	0.056
commented	57 714.0	0.0	0.886
committed	30 600.0	1.0	0.0
connected	30 600.0	1.0	0.0
convert_to_draft	30 600.0	1.0	0.0
created	57 031.5	0.0	0.864
cross_referenced	30 600.0	1.0	0.0
disconnected	30 600.0	1.0	0.0
head_ref_deleted	50 081.5	0.0	0.637
head_ref_force_pushed	56 871.0	0.0	0.859
head_ref_restored	33 928.0	0.0	0.109
mentioned	59 320.5	0.0	0.939
ready_for_review	30 573.0	0.551	−0.001
referenced_by	56 038.5	0.0	0.831
renamed	54 175.5	0.0	0.77
review_request_removed	32 291.5	0.0	0.055
review_requested	42 357.5	0.0	0.384
reviewed	53 999.0	0.0	0.765
state_updated	56 820.0	0.0	0.857
subscribed	55 540.5	0.0	0.815
unsubscribed	32 233.5	0.0	0.053

Table A.8: RQ1 VSCode

Variable	U	p-value	Cliff's Delta
Counted commits	1 789 967.5	0.0	0.703
Counted events	2 091 016.0	0.0	0.989
Mean of counted commits	1 789 095.0	0.0	0.702
Mean of counted events	2 026 100.5	0.0	0.927
assigned	1 491 719.5	0.0	0.419
demilestoned	1 186 879.5	0.0	0.129
labeled	1 423 750.0	0.0	0.354
marked_as_duplicate	1 085 185.0	0.0	0.032
milestoned	1 322 344.0	0.0	0.258
resolution_updated	1 220 770.0	0.0	0.161
type_updated	1 254 631.0	0.0	0.193
unassigned	1 423 327.5	0.0	0.354
unlabeled	1 356 355.0	0.0	0.29
unmarked_as_duplicate	1 051 287.5	1.0	0.0
add_link	1 485 403.0	0.0	0.413
commit_added	1 674 909.0	0.0	0.593
referenced	1 248 681.0	0.0	0.188
automatic_base_change_failed	1 051 287.5	1.0	0.0
automatic_base_change_succeeded	1 051 287.5	1.0	0.0
base_ref_changed	1 051 256.5	0.512	0.0
base_ref_force_pushed	1 051 287.5	1.0	0.0
comment_deleted	1 051 287.5	1.0	0.0
commented	1 867 159.0	0.0	0.776
committed	1 051 287.5	1.0	0.0
connected	1 051 287.5	1.0	0.0
convert_to_draft	1 051 163.5	0.524	0.0
created	1 756 657.5	0.0	0.671
cross_referenced	1 051 287.5	1.0	0.0
disconnected	1 051 287.5	1.0	0.0
head_ref_deleted	1 246 971.5	0.0	0.186
head_ref_force_pushed	1 216 049.5	0.0	0.157
head_ref_restored	1 084 786.5	0.0	0.032
mentioned	1 870 734.5	0.0	0.779
ready_for_review	1 050 636.5	0.555	−0.001
referenced_by	1 692 752.0	0.0	0.61
renamed	1 476 573.5	0.0	0.405
review_request_removed	1 051 256.5	0.512	0.0
review_requested	1 151 993.5	0.0	0.096
reviewed	1 249 392.0	0.0	0.188
state_updated	1 545 889.0	0.0	0.47
subscribed	1 797 178.0	0.0	0.71
unsubscribed	1 417 967.5	0.0	0.349

Table A.9: RQ1 Vue

Variable	U	p-value	Cliff's Delta
Counted commits	52 084.5	0.0	0.703
Counted events	60 216.5	0.0	0.969
Mean of counted commits	51 912.0	0.0	0.698
Mean of counted events	59 397.0	0.0	0.942
assigned	48 037.5	0.0	0.571
demilestoned	30 579.5	1.0	0.0
labeled	52 406.0	0.0	0.714
marked_as_duplicate	39 314.0	0.0	0.286
milestoned	30 579.5	1.0	0.0
resolution_updated	34 945.0	0.0	0.143
type_updated	52 401.0	0.0	0.714
unassigned	39 292.0	0.0	0.285
unlabeled	43 668.0	0.0	0.428
unmarked_as_duplicate	30 579.5	1.0	0.0
add_link	51 445.0	0.0	0.682
commit_added	51 512.0	0.0	0.685
referenced	47 679.0	0.0	0.559
automatic_base_change_failed	30 579.5	1.0	0.0
automatic_base_change_succeeded	30 579.5	1.0	0.0
base_ref_changed	30 576.0	0.513	0.0
base_ref_force_pushed	30 579.5	1.0	0.0
comment_deleted	30 579.5	1.0	0.0
commented	58 308.5	0.0	0.907
committed	30 579.5	1.0	0.0
connected	30 579.5	1.0	0.0
convert_to_draft	30 572.5	0.517	0.0
created	55 076.0	0.0	0.801
cross_referenced	30 579.5	1.0	0.0
disconnected	30 579.5	1.0	0.0
head_ref_deleted	43 395.5	0.0	0.419
head_ref_force_pushed	52 251.0	0.0	0.709
head_ref_restored	30 558.5	0.528	−0.001
mentioned	52 950.5	0.0	0.732
ready_for_review	30 565.5	0.523	0.0
referenced_by	51 475.0	0.0	0.683
renamed	50 447.0	0.0	0.65
review_request_removed	30 576.0	0.513	0.0
review_requested	39 254.5	0.0	0.284
reviewed	52 187.5	0.0	0.707
state_updated	51 821.0	0.0	0.695
subscribed	50 673.0	0.0	0.657
unsubscribed	30 471.0	0.563	−0.004

A.2 RESULTS FOR RQ2

Table A.10 to Table A.27 present the results of the one-tailed Wilcoxon signed rank tests for RQ2, comparing the behavior of privileged developers before and after receiving privileges with regard to each issue event and the number of commits and issue events. The Wilcoxon signed rank test evaluates for each variable whether there is a significant difference in the distributions of the two groups, focusing on whether the distribution of values in the first group (privileged developers before receiving privileges) tends to be less than that of the second group (privileged developers after receiving privileges). The W-statistic represents the test statistic, while the corresponding p-value indicates whether the alternative hypothesis is accepted ($p < 0.05$) or not. Additionally, R provides the corresponding effect size.

Table A.10: RQ2 Angular (common and extended events)

Variable	W	p-value	R
Counted commits	188.0	0.0	−0.685
Counted events	139.0	0.0	−0.769
Mean of counted commits	555.0	0.004	−0.342
Mean of counted events	768.0	0.003	−0.332
assigned	180.0	0.0	−0.709
demilestoned	98.0	0.0	−0.673
labeled	105.0	0.0	−0.781
marked_as_duplicate	0.0	0.001	−0.883
milestoned	123.5	0.0	−0.694
resolution_updated	0.0	0.5	−1.0
type_updated	4.0	0.104	−0.555
unassigned	160.5	0.0	−0.689
unlabeled	143.5	0.0	−0.746
unmarked_as_duplicate	0.0	1.0	0.0
add_link	186.0	0.0	−0.714
commit_added	127.0	0.0	−0.764
referenced	40.0	0.0	−0.755
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	16.0	0.001	−0.714
base_ref_force_pushed	0.0	0.011	−0.896
comment_deleted	0.0	0.5	−1.0
commented	169.5	0.0	−0.748
committed	0.0	1.0	0.0
connected	0.0	0.018	−0.898
convert_to_draft	0.0	0.001	−0.882
created	226.0	0.0	−0.696
cross_referenced	0.0	1.0	0.0
disconnected	0.0	0.5	−1.0
head_ref_deleted	196.0	0.0	−0.602
head_ref_force_pushed	162.5	0.0	−0.729
head_ref_restored	41.5	0.009	−0.53
mentioned	192.0	0.0	−0.723
ready_for_review	0.0	0.0	−0.873
referenced_by	233.0	0.0	−0.686
renamed	209.5	0.0	−0.641
review_request_removed	0.0	0.0	−0.871
review_requested	35.0	0.0	−0.823
reviewed	25.0	0.0	−0.837
state_updated	172.5	0.0	−0.738
subscribed	196.0	0.0	−0.72
unsubscribed	5.5	0.0	−0.838

Table A.11: RQ2 ANGULAR (privileged events)

Variable	W	p-value	R
added_to_project	0.0	0.0	−0.877
converted_note_to_issue	0.0	0.018	−0.898
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	0.5	−1.0
merged	0.0	0.0	−0.873
moved_columns_in_project	0.0	0.0	−0.874
pinned	0.0	0.018	−0.902
removed_from_project	0.0	0.001	−0.883
review_dismissed	0.0	0.0	−0.876
transferred	0.0	0.0	−0.878
unlocked	0.0	0.5	−1.0
unpinned	0.0	0.007	−0.956
user_blocked	0.0	1.0	0.0

Table A.12: RQ2 Atom (common and extended events)

Variable	W	p-value	R
Counted commits	78.0	0.0	−0.734
Counted events	110.5	0.0	−0.74
Mean of counted commits	282.5	0.007	−0.374
Mean of counted events	770.0	0.595	0.032
assigned	98.0	0.0	−0.6
demilestoned	0.0	1.0	0.0
labeled	61.0	0.0	−0.748
marked_as_duplicate	7.0	0.136	−0.45
milestoned	0.0	0.5	−1.0
resolution_updated	17.0	0.001	−0.736
type_updated	148.0	0.009	−0.413
unassigned	32.5	0.002	−0.631
unlabeled	116.0	0.005	−0.465
unmarked_as_duplicate	0.0	0.5	−1.0
add_link	163.5	0.0	−0.648
commit_added	69.5	0.0	−0.768
referenced	89.5	0.015	−0.428
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	0.091	−0.924
base_ref_force_pushed	0.0	0.03	−0.903
comment_deleted	0.0	1.0	0.0
commented	275.0	0.0	−0.548
committed	0.0	1.0	0.0
connected	0.0	1.0	0.0
convert_to_draft	0.0	0.5	−1.0
created	236.0	0.0	−0.573
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	14.5	0.0	−0.847
head_ref_force_pushed	106.5	0.001	−0.561
head_ref_restored	0.0	0.0	−0.88
mentioned	291.5	0.0	−0.487
ready_for_review	0.0	0.005	−0.89
referenced_by	152.0	0.0	−0.671
renamed	117.5	0.0	−0.645
review_request_removed	0.0	0.018	−0.898
review_requested	5.0	0.0	−0.842
reviewed	35.5	0.0	−0.72
state_updated	75.5	0.0	−0.763
subscribed	223.5	0.0	−0.577
unsubscribed	0.0	0.003	−0.889

Table A.13: RQ2 Atom (privileged events)

Variable	W	p-value	R
added_to_project	0.0	1.0	0.0
converted_note_to_issue	0.0	1.0	0.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	1.0	0.0
merged	0.0	0.0	−0.871
moved_columns_in_project	0.0	1.0	0.0
pinned	0.0	0.5	−1.0
removed_from_project	0.0	1.0	0.0
review_dismissed	0.0	0.091	−0.924
transferred	0.0	0.049	−0.92
unlocked	0.0	1.0	0.0
unpinned	0.0	0.186	−0.948
user_blocked	0.0	1.0	0.0

Table A.14: RQ2 Bootstrap (common and extended events)

Variable	W	p-value	R
Counted commits	12.5	0.004	−0.697
Counted events	4.0	0.0	−0.837
Mean of counted commits	61.0	0.534	0.015
Mean of counted events	97.0	0.695	0.118
assigned	0.0	0.003	−0.885
demilestoned	0.0	0.003	−0.885
labeled	1.0	0.0	−0.865
marked_as_duplicate	0.0	0.011	−0.896
milestoned	0.0	0.003	−0.885
resolution_updated	0.0	0.05	−0.915
type_updated	0.0	1.0	0.0
unassigned	0.0	0.011	−0.896
unlabeled	0.0	0.001	−0.882
unmarked_as_duplicate	0.0	1.0	0.0
add_link	34.0	0.023	−0.487
commit_added	46.0	0.078	−0.349
referenced	18.0	0.179	−0.306
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	0.03	−0.903
base_ref_force_pushed	0.0	0.03	−0.903
comment_deleted	0.0	0.5	−1.0
commented	35.0	0.026	−0.475
committed	0.0	1.0	0.0
connected	0.0	0.091	−0.924
convert_to_draft	0.0	0.03	−0.903
created	50.0	0.109	−0.303
cross_referenced	0.0	1.0	0.0
disconnected	0.0	0.5	−1.0
head_ref_deleted	33.0	0.116	−0.326
head_ref_force_pushed	18.0	0.016	−0.58
head_ref_restored	15.0	0.6	0.064
mentioned	32.0	0.019	−0.512
ready_for_review	1.0	0.01	−0.841
referenced_by	30.0	0.015	−0.534
renamed	22.0	0.03	−0.51
review_request_removed	0.0	0.05	−0.915
review_requested	3.0	0.003	−0.814
reviewed	7.0	0.004	−0.746
state_updated	9.0	0.001	−0.762
subscribed	31.0	0.008	−0.559
unsubscribed	0.0	0.03	−0.903

Table A.15: RQ2 Bootstrap (privileged events)

Variable	W	p-value	R
added_to_project	0.0	0.005	−0.89
converted_note_to_issue	0.0	0.5	−1.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	0.5	−1.0
merged	0.0	0.001	−0.882
moved_columns_in_project	0.0	0.005	−0.89
pinned	0.0	0.091	−0.924
removed_from_project	0.0	0.007	−0.891
review_dismissed	0.0	0.018	−0.898
transferred	0.0	1.0	0.0
unlocked	0.0	0.5	−1.0
unpinned	0.0	0.091	−0.924
user_blocked	0.0	0.5	−1.0

Table A.16: RQ2 Deno (common and extended events)

Variable	W	p-value	R
Counted commits	6.0	0.102	−0.51
Counted events	6.5	0.062	−0.569
Mean of counted commits	5.0	0.075	−0.575
Mean of counted events	7.0	0.074	−0.544
assigned	0.0	0.05	−0.915
demilestoned	0.0	0.091	−0.924
labeled	0.0	0.05	−0.915
marked_as_duplicate	0.0	0.186	−0.948
milestoned	0.0	0.091	−0.924
resolution_updated	0.0	0.186	−0.948
type_updated	0.0	0.091	−0.924
unassigned	0.0	0.091	−0.924
unlabeled	0.0	0.091	−0.924
unmarked_as_duplicate	0.0	1.0	0.0
add_link	6.0	0.102	−0.51
commit_added	5.0	0.075	−0.575
referenced	0.0	0.186	−0.948
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	0.186	−0.948
base_ref_force_pushed	0.0	0.186	−0.948
comment_deleted	0.0	0.5	−1.0
commented	6.0	0.102	−0.51
committed	0.0	1.0	0.0
connected	0.0	1.0	0.0
convert_to_draft	1.0	0.5	−0.316
created	6.0	0.102	−0.51
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	2.0	0.047	−0.727
head_ref_force_pushed	4.0	0.104	−0.555
head_ref_restored	0.0	0.186	−0.948
mentioned	6.0	0.102	−0.51
ready_for_review	8.0	0.606	0.06
referenced_by	6.0	0.102	−0.51
renamed	3.0	0.071	−0.641
review_request_removed	0.0	0.05	−0.915
review_requested	1.0	0.053	−0.783
reviewed	4.0	0.054	−0.639
state_updated	2.0	0.047	−0.727
subscribed	6.0	0.102	−0.51
unsubscribed	0.0	0.091	−0.924

Table A.17: RQ2 DENO (privileged events)

Variable	W	p-value	R
added_to_project	0.0	1.0	0.0
converted_note_to_issue	0.0	1.0	0.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	0.5	−1.0
merged	0.0	0.05	−0.915
moved_columns_in_project	0.0	1.0	0.0
pinned	0.0	0.186	−0.948
removed_from_project	0.0	1.0	0.0
review_dismissed	0.0	0.091	−0.924
transferred	0.0	0.186	−0.948
unlocked	0.0	0.5	−1.0
unpinned	0.0	0.186	−0.948
user_blocked	0.0	1.0	0.0

Table A.18: RQ2 Electron (common and extended events)

Variable	W	p-value	R
Counted commits	149.0	0.027	−0.348
Counted events	141.0	0.0	−0.599
Mean of counted commits	268.0	0.656	0.07
Mean of counted events	400.0	0.264	−0.099
assigned	33.5	0.0	−0.694
demilestoned	0.0	1.0	0.0
labeled	90.0	0.0	−0.592
marked_as_duplicate	2.0	0.047	−0.727
milestoned	0.0	1.0	0.0
resolution_updated	30.0	0.147	−0.3
type_updated	75.5	0.491	−0.011
unassigned	27.0	0.003	−0.629
unlabeled	66.0	0.002	−0.57
unmarked_as_duplicate	0.0	0.5	−1.0
add_link	139.5	0.004	−0.463
commit_added	68.0	0.0	−0.661
referenced	40.5	0.026	−0.462
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	0.5	−1.0
base_ref_changed	8.0	0.002	−0.762
base_ref_force_pushed	14.0	0.312	−0.198
comment_deleted	0.0	0.5	−1.0
commented	128.5	0.0	−0.584
committed	0.0	1.0	0.0
connected	0.0	1.0	0.0
convert_to_draft	0.0	0.05	−0.915
created	130.5	0.0	−0.565
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	49.0	0.0	−0.676
head_ref_force_pushed	33.0	0.0	−0.71
head_ref_restored	3.0	0.004	−0.805
mentioned	215.0	0.012	−0.367
ready_for_review	3.0	0.003	−0.814
referenced_by	127.5	0.001	−0.538
renamed	66.5	0.0	−0.665
review_request_removed	22.0	0.098	−0.384
review_requested	30.0	0.0	−0.745
reviewed	26.0	0.0	−0.781
state_updated	129.0	0.0	−0.568
subscribed	163.0	0.001	−0.508
unsubscribed	0.0	0.005	−0.89

Table A.19: RQ2 ELECTRON (privileged events)

Variable	W	p-value	R
added_to_project	0.0	0.001	−0.883
converted_note_to_issue	0.0	0.186	−0.948
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	0.5	−1.0
merged	0.0	0.0	−0.875
moved_columns_in_project	0.0	0.001	−0.883
pinned	0.0	0.05	−0.915
removed_from_project	0.0	0.005	−0.89
review_dismissed	0.0	0.003	−0.885
transferred	0.0	0.186	−0.948
unlocked	0.0	0.5	−1.0
unpinned	0.0	0.087	−0.941
user_blocked	0.0	1.0	0.0

Table A.20: RQ2 KERAS (common and extended events)

Variable	W	p-value	R
Counted commits	150.5	0.117	−0.227
Counted events	279.5	0.496	−0.003
Mean of counted commits	216.0	0.371	−0.062
Mean of counted events	302.5	0.656	0.068
assigned	5.0	0.147	−0.469
demilestoned	0.0	1.0	0.0
labeled	0.0	0.05	−0.915
marked_as_duplicate	0.0	1.0	0.0
milestoned	0.0	1.0	0.0
resolution_updated	0.0	0.186	−0.948
type_updated	0.0	0.091	−0.924
unassigned	3.0	0.605	0.0
unlabeled	0.0	0.05	−0.915
unmarked_as_duplicate	0.0	1.0	0.0
add_link	125.0	0.16	−0.202
commit_added	153.5	0.292	−0.11
referenced	1.0	0.211	−0.618
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	1.0	0.0
base_ref_force_pushed	0.0	1.0	0.0
comment_deleted	0.0	1.0	0.0
commented	221.5	0.305	−0.093
committed	0.0	1.0	0.0
connected	0.0	1.0	0.0
convert_to_draft	0.0	1.0	0.0
created	230.0	0.611	0.05
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	79.5	0.273	−0.143
head_ref_force_pushed	25.5	0.661	0.119
head_ref_restored	11.0	0.86	0.422
mentioned	147.0	0.472	−0.018
ready_for_review	0.0	0.5	−1.0
referenced_by	91.0	0.128	−0.245
renamed	83.0	0.974	0.513
review_request_removed	0.0	0.186	−0.948
review_requested	0.0	0.018	−0.898
reviewed	27.5	0.062	−0.42
state_updated	89.5	0.42	−0.051
subscribed	147.0	0.472	−0.018
unsubscribed	4.0	0.428	−0.182

Table A.21: RQ2 KERAS (privileged events)

Variable	W	p-value	R
added_to_project	0.0	1.0	0.0
converted_note_to_issue	0.0	0.5	−1.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	1.0	0.0
merged	0.0	0.03	−0.903
moved_columns_in_project	0.0	0.091	−0.924
pinned	0.0	1.0	0.0
removed_from_project	0.0	1.0	0.0
review_dismissed	0.0	0.018	−0.902
transferred	0.0	1.0	0.0
unlocked	0.0	1.0	0.0
unpinned	0.0	1.0	0.0
user_blocked	0.0	1.0	0.0

Table A.22: RQ2 OpenSSL (common and extended events)

Variable	W	p-value	R
Counted commits	119.0	0.931	0.344
Counted events	58.5	0.074	−0.337
Mean of counted commits	88.0	0.552	0.026
Mean of counted events	25.0	0.002	−0.647
assigned	23.0	0.011	−0.582
demilestoned	12.0	0.063	−0.5
labeled	5.0	0.001	−0.815
marked_as_duplicate	1.0	0.211	−0.618
milestoned	16.0	0.071	−0.455
resolution_updated	0.0	0.091	−0.924
type_updated	1.0	0.002	−0.859
unassigned	3.0	0.012	−0.77
unlabeled	8.0	0.001	−0.775
unmarked_as_duplicate	0.0	1.0	0.0
add_link	12.5	0.001	−0.75
commit_added	27.0	0.006	−0.601
referenced	3.0	0.605	0.0
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	0.5	−1.0
base_ref_force_pushed	0.0	0.5	−1.0
comment_deleted	0.0	0.5	−1.0
commented	17.0	0.002	−0.702
committed	0.0	1.0	0.0
connected	0.0	1.0	0.0
convert_to_draft	0.0	1.0	0.0
created	37.0	0.018	−0.497
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	26.0	0.051	−0.444
head_ref_force_pushed	37.0	0.057	−0.4
head_ref_restored	0.0	0.03	−0.903
mentioned	25.0	0.004	−0.62
ready_for_review	0.0	0.091	−0.924
referenced_by	11.0	0.001	−0.752
renamed	12.0	0.002	−0.725
review_request_removed	2.0	0.026	−0.767
review_requested	9.0	0.002	−0.749
reviewed	11.0	0.002	−0.738
state_updated	21.0	0.003	−0.662
subscribed	16.0	0.002	−0.694
unsubscribed	1.0	0.211	−0.618

Table A.23: RQ2 OpenSSL (privileged events)

Variable	W	p-value	R
added_to_project	0.0	0.018	−0.898
converted_note_to_issue	0.0	0.5	−1.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	1.0	0.0
merged	0.0	0.091	−0.924
moved_columns_in_project	0.0	0.002	−0.883
pinned	0.0	0.5	−1.0
removed_from_project	0.0	0.186	−0.948
review_dismissed	0.0	0.003	−0.885
transferred	0.0	1.0	0.0
unlocked	0.0	1.0	0.0
unpinned	0.0	0.5	−1.0
user_blocked	0.0	0.5	−1.0

Table A.24: RQ2 VSCode (common and extended events)

Variable	W	p-value	R
Counted commits	30.5	0.0	−0.82
Counted events	61.5	0.0	−0.786
Mean of counted commits	264.0	0.001	−0.445
Mean of counted events	452.0	0.037	−0.253
assigned	31.0	0.0	−0.818
demilestoned	7.0	0.0	−0.854
labeled	47.0	0.0	−0.788
marked_as_duplicate	10.0	0.0	−0.824
milestoned	13.0	0.0	−0.842
resolution_updated	19.0	0.0	−0.81
type_updated	18.0	0.0	−0.829
unassigned	19.0	0.0	−0.838
unlabeled	45.0	0.0	−0.788
unmarked_as_duplicate	0.0	0.186	−0.948
add_link	93.0	0.0	−0.733
commit_added	65.0	0.0	−0.752
referenced	48.0	0.0	−0.694
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	0.003	−0.885
base_ref_force_pushed	0.0	1.0	0.0
comment_deleted	0.0	0.5	−1.0
commented	148.0	0.0	−0.66
committed	0.0	1.0	0.0
connected	0.0	0.03	−0.903
convert_to_draft	0.0	0.049	−0.92
created	120.5	0.0	−0.684
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	6.0	0.0	−0.855
head_ref_force_pushed	9.0	0.0	−0.823
head_ref_restored	13.0	0.007	−0.663
mentioned	142.0	0.0	−0.676
ready_for_review	0.0	0.001	−0.882
referenced_by	74.0	0.0	−0.761
renamed	65.0	0.0	−0.752
review_request_removed	0.0	0.0	−0.874
review_requested	17.0	0.0	−0.834
reviewed	9.5	0.0	−0.845
state_updated	50.0	0.0	−0.787
subscribed	118.0	0.0	−0.696
unsubscribed	71.0	0.0	−0.74

Table A.25: RQ2 VSCode (privileged events)

Variable	W	p-value	R
added_to_project	0.0	0.186	−0.948
converted_note_to_issue	0.0	1.0	0.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	0.5	−1.0
merged	0.0	0.0	−0.871
moved_columns_in_project	0.0	0.186	−0.948
pinned	0.0	0.0	−0.876
removed_from_project	0.0	0.5	−1.0
review_dismissed	0.0	0.011	−0.9
transferred	0.0	0.0	−0.876
unlocked	0.0	0.5	−1.0
unpinned	0.0	0.0	−0.886
user_blocked	0.0	1.0	0.0

Table A.26: RQ2 Vue (common and extended events)

Variable	W	p-value	R
Counted commits	28.5	0.361	−0.121
Counted events	48.0	0.404	−0.075
Mean of counted commits	39.0	0.718	0.161
Mean of counted events	54.0	0.548	0.025
assigned	28.0	0.761	0.217
demilestoned	0.0	0.5	−1.0
labeled	35.0	0.588	0.054
marked_as_duplicate	3.0	0.292	−0.365
milestoned	0.0	0.5	−1.0
resolution_updated	2.0	0.395	−0.309
type_updated	28.0	0.541	0.016
unassigned	7.0	0.819	0.365
unlabeled	9.0	0.417	−0.128
unmarked_as_duplicate	0.0	1.0	0.0
add_link	40.0	0.547	0.023
commit_added	36.0	0.422	−0.068
referenced	21.0	0.688	0.148
automatic_base_change_failed	0.0	1.0	0.0
automatic_base_change_succeeded	0.0	1.0	0.0
base_ref_changed	0.0	0.186	−0.948
base_ref_force_pushed	0.0	0.5	−1.0
comment_deleted	0.0	0.5	−1.0
commented	48.0	0.583	0.049
committed	0.0	1.0	0.0
connected	0.0	1.0	0.0
convert_to_draft	0.0	1.0	0.0
created	54.0	0.735	0.165
cross_referenced	0.0	1.0	0.0
disconnected	0.0	1.0	0.0
head_ref_deleted	16.0	0.131	−0.37
head_ref_force_pushed	41.0	0.775	0.215
head_ref_restored	0.0	0.087	−0.941
mentioned	39.0	0.337	−0.126
ready_for_review	0.0	1.0	0.0
referenced_by	40.0	0.547	0.023
renamed	26.5	0.48	−0.032
review_request_removed	0.0	0.5	−1.0
review_requested	8.0	0.176	−0.382
reviewed	26.0	0.282	−0.188
state_updated	35.0	0.392	−0.091
subscribed	35.0	0.242	−0.204
unsubscribed	0.0	0.186	−0.948

Table A.27: RQ2 VUE (privileged events)

Variable	W	p-value	R
added_to_project	0.0	0.5	−1.0
converted_note_to_issue	0.0	1.0	0.0
deployed	0.0	1.0	0.0
deployment_environment_changed	0.0	1.0	0.0
locked	0.0	0.5	−1.0
merged	0.0	0.091	−0.924
moved_columns_in_project	0.0	0.5	−1.0
pinned	0.0	1.0	0.0
removed_from_project	0.0	0.5	−1.0
review_dismissed	0.0	0.5	−1.0
transferred	0.0	0.5	−1.0
unlocked	0.0	0.5	−1.0
unpinned	0.0	1.0	0.0
user_blocked	0.0	1.0	0.0

A.3 RESULTS FOR RQ3

Table A.28 presents the results of the one-tailed Wilcoxon signed rank tests for RQ3, comparing the centralities of privileged developers before and after receiving privileges in issue networks. This test evaluates whether there is a significant difference in the distributions of the two groups, focusing on whether the distribution of values in the first group (privileged developers before receiving privileges) tends to be less than that of the second group (privileged developers after receiving privileges).

Similarly, Table A.29 presents the results of the two-tailed Wilcoxon signed rank tests for RQ3, comparing the centralities of privileged developers before and after receiving privileges in cochange networks. This test evaluates whether there is a significant difference in the distributions of centralities before and after receiving privileges, without specifying a particular trend of the difference.

Table A.28: RQ3 Issue networks

Project	W	p-value	R
`Angular`	215.0	0.0	−0.718
Atom	128.0	0.0	−0.72
Bootstrap	30.0	0.007	−0.57
Deno	6.0	0.055	−0.594
Electron	295.0	0.025	−0.302
Keras	176.0	0.125	−0.212
OpenSSL	68.0	0.147	−0.25
VSCode	137.0	0.0	−0.683
Vue	39.0	0.213	−0.226

Table A.29: RQ3 Cochange networks

Project	W	p-value	R
`Angular`	256.0	0.0	−0.626
Atom	92.0	0.0	−0.709
Bootstrap	36.0	0.188	−0.351
Deno	7.0	0.297	−0.446
Electron	154.0	0.109	−0.294
Keras	113.0	0.303	−0.216
OpenSSL	79.0	0.799	−0.067
VSCode	51.0	0.0	−0.778
Vue	31.0	0.898	−0.054

BIBLIOGRAPHY

[1] Bram Adams, Ryan Kavanagh, and Ahmed E. Hassan. "An empirical study of integration activities in distributions of open source software." In: *Empirical Software Engineering* 21 (2016), pp. 960–1001.

[2] Christian Bird. "Sociotechnical coordination and collaboration in open source software." PhD thesis. 2010.

[3] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. "Latent social structure in open source projects." In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2008, 24–35.

[4] Thomas Bock, Nils Alznauer, Mitchell Joblin, and Sven Apel. "Automatic Core-Developer Identification on GitHub: A Validation Study." In: *ACM Transactions on Software Engineering and Methodology* 32.138 (6 2023), pp. 1–29.

[5] Phillip Bonacich. "Factoring and weighting approaches to status scores and clique identification." In: *Journal of Mathematical Sociology* 2 (1972), pp. 113–120.

[6] Phillip Bonacich. "Some unique properties of eigenvector centrality." In: *Social Networks* 29.4 (2007), pp. 555–564.

[7] Stephen P. Borgatti, Martin G. Everett, and Jeffrey C. Johnson. *Analyzing Social Networks*. 2nd. Sage, 2018.

[8] Ulrik Brandes and Thomas Erlebach, eds. *Network Analysis. Methodological Foundations*. Lecture Notes in Computer Science. Springer Berlin, Heidelberg, 2005.

[9] Chris Brown and Chris Parnin. "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] Edna D. Canedo, Rodrigo Bonifácio, Márcio V. Okimoto, Alexander Serebrenik, Gustavo Pinto, and Eduardo Monteiro. "Work Practices and Perceptions from Women Core Developers in OSS Communities." In: *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2020, pp. 1–11.

[11] Marcelo Cataldo and James D. Herbsleb. "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures." In: *IEEE Transactions on Software Engineering* 39.3 (2013), pp. 343–360.

[12] Kevin Crowston and James Howison. "The social structure of free and open source software development." In: *First Monday* 10 (2005).

[13] Kevin Crowston, Qing Li, Kangning Wei, U. Yeliz Eseryel, and James Howison. "Self-Organization of Teams for Free/Libre Open Source Software Development." In: *Information and Software Technology (IST)* 49.6 (2007), 564–575.

[14] Kewin Crowston, Kangning Wei, Qing Li, and J. 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. 2006, 118a–118a.

[15] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. "What Makes a Great Maintainer of Open Source Projects?" In: *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021, pp. 982–994.

[16] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, and Vladimir Filkov. "Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation." In: *Empirical Software Engineering* 20.5 (2015), 1318–1353.

[17] GitHub. *About commits*. URL: https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits (visited on 03/21/2024).

[18] GitHub. *Automatically merging a pull request*. URL: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/automatically-merging-a-pull-request (visited on 03/21/2024).

[19] GitHub. *GitHub Issue Event Types*. URL: https://docs.github.com/en/rest/using-the-rest-api/issue-event-types (visited on 03/15/2024).

[20] GitHub. *Issues*. URL: https://docs.github.com/en/issues (visited on 03/21/2024).

[21] GitHub. *Pull requests*. URL: https://docs.github.com/en/pull-requests (visited on 03/21/2024).

[22] Mehdi Golzadeh, Damien Legay, Alexandre Decan, and Tom Mens. "Bot or not? Detecting bots in GitHub pull request activity based on comment similarity." In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 2020, 31–35.

[23] Rebecca Grinter, James Herbsleb, and Dewayne Perry. "The geography of coordination: Dealing with distance in R and D work." In: Nov. 1999, pp. 306–315.

[24] James D. Herbsleb and Jeffrey A. Roberts. "Collaboration In Software Engineering Projects: A Theory Of Coordination." In: *International Conference on Interaction Sciences*. 2006.

[25] Qiaona Hong, Sunghun Kim, S.C. Cheung, and Christian Bird. "Understanding a developer social network and its evolution." In: *27th IEEE International Conference on Software Maintenance (ICSM)*. 2011, pp. 323–332.

[26] Open Source Initiative. *Open Source Definition*. Open Source Initiative. URL: https://opensource.org/osd (visited on 03/25/2024).

[27] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. "The Onion Patch: Migration in Open Source Ecosystems." In: *Proceedings of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2011, pp. 70–80.

[28] 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 International Conference on Software Engineering (ICSE)*.

[29] Mitchell Joblin, Sven Apel, and Wolfgang Mauerer. "Evolutionary Trends of Developer Coordination: A Network Approach." In: *Empirical Software Engineering* 22.4 (2017), 2050–2094.

[30] Mitchell Joblin, Barbara Eckl-Ganser, Thomas Bock, Angelika Schmid, Janet Siegmund, and Sven Apel. "Hierarchical and Hybrid Organizational Structures in Open-Source Software Projects: A Longitudinal Study." In: *ACM Transactions on Software Engineering and Methodology* 32.86 (4 2023), pp. 1–29.

[31] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. "From developer networks to verified communities: a fine-grained approach." In: *Proceedings of the 37th International Conference on Software Engineering*. Vol. 1. 2015, 563–573.

[32] Robert E. Kraut and Lynn A. Streeter. "Coordination in Software Development." In: *Communications of the ACM* 38.3 (1995), 69–81.

[33] Irwin Kwan, Adrian Schroter, and Daniela Damian. "Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project." In: *IEEE Transactions on Software Engineering* 37.3 (2011), pp. 307–324.

[34] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. "Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey." In: *IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 187–197.

[35] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. "Two Case Studies of Open Source Software Development: Apache and Mozilla." In: *ACM Transactions on Software Engineering and Methodology* 11.3 (2002), 309–346.

[36] CRAN Task View: Psychometric Models and Methods. *R: Cliff's Delta*. URL: https://search.r-project.org/CRAN/refmans/rcompanion/html/cliffDelta.html (visited on 03/15/2024).

[37] CRAN Task View: Psychometric Models and Methods. *R: wilcoxonPairedR()*. URL: https://www.rdocumentation.org/packages/rcompanion/versions/2.3.7/topics/wilcoxonPairedR (visited on 03/15/2024).

[38] 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 (IWPSE)*. 2002.

[39] Sebastiano Panichella, Gabriele Bavota, Massimiliano Di Penta, Gerardo Canfora, and Giuliano Antoniol. "How Developers' Collaborations Identified from Different Sources Tell Us about Code Changes." In: *IEEE International Conference on Software Maintenance and Evolution*. 2014, pp. 251–260.

[40] Gustavo Pinto, Igor Steinmacher, and Marco A. Gerosa. "More Common than You Think: An In-Depth Study of Casual Contributors." In: *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2016.

[41] Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, and Audris Mockus. "Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a Project at Avaya." In: *Proceedings of the International Conference on Software Engineering (ICSE)*.

[42] Pankaj Setia, Balaji Rajagopalan, Vallabh Sambamurthy, and Roger Calantone. "How Peripheral Developers Contribute to Open-Source Software Development." In: *Information Systems Research* 23.1 (2012), 144–163.

[43] Damian A. Tamburri, Patricia Lago, and Hans van Vliet. "Organizational social structures for software engineering." In: *ACM Computing Surveys* 46.1 (2013).

[44] Antonio Terceiro, Luiz Romario Rios, and Christina Chavez. "An Empirical Study on the Structural Complexity Introduced by Core and Peripheral Developers in Free Software Projects." In: *Brazilian Symposium on Software Engineering*. 2010, pp. 21–29.

[45] Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. "Unveiling Elite Developers' Activities in Open Source Projects." In: 29.3 (2020).

[46] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[47] 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: *Proceedings of the ACM Human-Computer Interaction* 2.CSCW (2018).

[48] 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*. 2020, 51–55.

[49] Marvin Wyrich and Justus Bogner. "Towards an Autonomous Bot for Automatic Source Code Refactoring." In: *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)* (2019), pp. 24–28.

[50] Yunwen Ye and K. Kishida. "Toward an understanding of the motivation of open source software developers." In: *Proceedings of the 25th International Conference on Software Engineering*. 2003, pp. 419–429.

[51] Liguo Yu. "Self-organization process in open-source software: An empirical study." In: *Information and Software Technology (IST)* 50.5 (2008), pp. 361–374.

[52] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. "On the scalability of Linux kernel maintainers' work." In: Aug. 2017, pp. 27–37.

[53] Minghui Zhou and Audris Mockus. "What make long term contributors: Willingness and opportunity in OSS community." In: *34th International Conference on Software Engineering (ICSE)*. 2012, pp. 518–528.