Core Developer Turnover in the Rust Package Ecosystem: Prevalence, Impact, and Awareness

MENG FAN, Beijing Institute of Technology, China
YUXIA ZHANG*, Beijing Institute of Technology, China
KLAAS-JAN STOL, University College Cork, Ireland, Lero, Ireland, and SINTEF, Norway
HUI LIU, Beijing Institute of Technology, China

Continued contributions of core developers in open source software (OSS) projects are key for sustaining and maintaining successful OSS projects. A major risk to the sustainability of OSS projects is developer turnover. Prior studies have explored developer turnover at the level of individual projects. A shortcoming of such studies is that they ignore the impact of developer turnover on downstream projects. Yet, an awareness of the turnover of core developers offers useful insights to the rest of an open source ecosystem. This study performs a large-scale empirical analysis of code developer turnover in the Rust package ecosystem. We find that the turnover of core developers is quite common in the whole Rust ecosystem with 36,991 packages. This is particularly worrying as a vast majority of Rust packages only have a single core developer. We found that core developer turnover can significantly decrease the quality and efficiency of software development and maintenance, even leading to deprecation. This is a major source of concern for those Rust packages that are widely used. We surveyed developers' perspectives on the turnover of core developers in upstream packages. We found that developers widely agreed that core developer turnover can affect project stability and sustainability. They also emphasized the importance of transparency and timely notifications regarding the health status of upstream dependencies. This study provides unique insights to help communities focus on building reliable software dependency networks.

CCS Concepts: • **Software and its engineering → Collaboration in software development**; **Open source model**; **Programming teams**.

Additional Key Words and Phrases: Open source software, core developer turnover, software dependency

ACM Reference Format:
Meng Fan, Yuxia Zhang, Klaas-Jan Stol, and Hui Liu. 2025. Core Developer Turnover in the Rust Package Ecosystem: Prevalence, Impact, and Awareness. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE122 (July 2025), 23 pages. https://doi.org/10.1145/3729392

1 Introduction

Open source software (OSS) projects are commonly used as libraries in product development. A recent report from Synopsys Black Duck [48] shows that 96% of 1,067 commercial codebases across 17 industries scanned in 2023 contain open source code. OSS has become the foundation of the digital world. Given society's dependence on OSS, it is key to develop an understanding of how OSS projects can succeed, and what barriers might exist that lead them to failure. Some research

*Corresponding author

Authors' Contact Information: Meng Fan, Beijing Institute of Technology, Beijing, China, fanmeng@bit.edu.cn; Yuxia Zhang, Beijing Institute of Technology, Beijing, China, yuxiazh@bit.edu.cn; Klaas-Jan Stol, University College Cork, Cork, Ireland and Lero, Limerick, Ireland and SINTEF, Trondheim, Norway, k.stol@ucc.ie; Hui Liu, Beijing Institute of Technology, Beijing, China, liuhui08@bit.edu.cn.

ACM 2994-970X/2025/7-ARTFSE122
https://doi.org/10.1145/3729392

suggests that most OSS projects fail [12], and developer turnover is one of the most common reasons [12]. Developers may leave an OSS project for various reasons, such as losing interest, lack of time [12, 56], career transitions [31], or financial issues [19]. Prior research found that the Pareto principle which has been frequently encountered in software engineering phenomena, also applies to OSS development [10, 14, 58]. That is, a small core group does most of the work and coordinates a much larger group of peripheral participants. Core developers play a significant role in setting the roadmap of OSS projects, accomplishing complex and critical tasks, and guiding newcomers [29]. The turnover of core developers can be a major setback for OSS projects, which poses a risk for their sustainability [7]. This threat coexists with the convenience of reusing third-party open source libraries [52]. For instance, vulnerabilities in a project that lost its core developers may not be fixed in a timely manner, which may lead to catastrophic consequences given that some projects have many dependent downstream projects [52].

The scale of this issue has, however, not been documented before. In this study, we make a first attempt with a specific focus on the Rust ecosystem. Rust is a popular open source programming language with an emphasis on performance, reliability, and productivity, with a rapidly growing package ecosystem [24]. We selected the Rust ecosystem as a case study because 1) Rust has been the top choice for developers who want to use new technology for the past eight years [47]; and 2) prior research on software dependencies is largely focused on the Python, JavaScript, and Java ecosystems. This paper therefore contributes a novel study of a highly popular ecosystem that has hitherto not been studied. This study makes a number of contributions, guided by four research questions. We start by asking:

RQ1: How prevalent is core developer turnover among Rust packages?

We answer RQ1 by analyzing the commonness of Rust packages that have lost their core developers, lost the top developer, and lost all their core developers. We find core developer turnover is quite common in the Rust packages. Over 77% of almost 37,000 Rust packages only have a single core developer; 56% of these packages have experienced the turnover of the sole core developer, rendering most of these packages abandoned. For the remaining 8,348 packages with more than one core developer, the turnover of core developers is also prevalent: over 73% of these packages have lost at least one core developer; 43% have lost their top developer; and over a quarter (27%) have lost all of their core developers.

Second, having established core developer turnover as a prevalent issue, we next examine the impact of this on many downstream projects. We ask:

RQ2: What impact does core developer turnover have on the packages they leave? We measure the impact of core developer turnover in the Rust ecosystem based on three key OSS activities: (1) submitting code changes; (2) code review; and (3) issue tracking. The results indicate that the turnover of core developers significantly affects packages with only one core developer. After the sole core developer leaves, the number of commits decreases markedly, reducing development and maintenance efficiency. Over 85% of packages cease to be maintained. The number of pull requests and issues declines, while the time to process pull requests and the ratio of unresolved issues increases. For packages with multiple core developers, a similar situation arises after the departure of the most active developer. Projects that have multiple core developers are more resilient than single-developer projects.

Third, we examine the scale of the issue in terms of the number of downstream packages affected. We ask: **RQ3: How many downstream packages are affected by the turnover of upstream core developers?** We identify the number of packages among the 36,991 that experienced three different types of core developer departure: top developer departure, complete core team departure, and maintenance cessation. We find that the departure of core developers impacts downstream software packages through software dependencies. Among the 113,319 (out of 149,445) packages

that have upstream dependencies, 61.2%, 45.9%, and 35.7% of the packages are directly affected by the departure of the top developer, the complete loss of core developers, and the discontinuation of library maintenance from upstream, respectively. If the indirect impact of dependencies on the abandoned package is taken into account, the situation becomes even more severe.

Finally, we investigate the perceptions of developers in downstream projects that have been affected by core developer turnover. We ask:

RQ4: How do developers perceive depending on packages that have lost core developers? We conduct email surveys to package owners who have public emails and inquire about their perspectives on the importance of core developers, their response to upstream's core developer turnover, and their views on receiving timely notifications about such departures. Then we analyze the open-ended questions in the survey using thematic analysis. We find that most respondents recognize the importance of core developers and are open to receiving timely notifications regarding the status of core developers in their upstream dependencies.

In sum, this study makes a number of contributions. First, this study is the first to focus on the entire Rust package ecosystem, with a specific focus on the prevalence of core developer departure. Second, this study provides a comprehensive measure of the impact of core developer departure in the Rust ecosystem based on three key activities in OSS development: submitting code changes, code review, and issue tracking. Third, this study is the first to focus on the software supply chain to explore the impact of core developer departure on downstream projects and convey downstream developers' perspectives towards upstream core developer departure.

2 Dataset

2.1 Data Collection

2.1.1 Rust Package Metadata. Crate.io [2] is the official package registry for the Rust programming language, providing a centralized platform for Rust developers to share and reuse software packages, called *'crates.'* It has not only facilitated the rapid growth of the Rust ecosystem but has also become an indispensable resource within the Rust community. The goal of Crate.io is to simplify the process of discovering, using, and maintaining packages while ensuring their security and reliability. To support research and data analysis, Crate.io offers a regularly updated database replica containing all packages and associated metadata in the registry. These database exports contain rich information, including package name, version, author, dependencies, and download counts. As of June 27, 2024, Crate.io has published a total of 149,445 software packages. We have extracted the names of these packages, the URLs of their corresponding code repositories, and their interdependencies.

2.1.2 Corresponding Rust Packages to GitHub Repositories. Each software package has a 'repository' entry corresponding to code repository platforms. Of the 149,445 packages, 28,426 do not have a code repository link provided, and after excluding these packages, 121,019 packages remained. By matching the pattern "https://github.com/<owner>/<name>", we removed 7,506 packages with repositories on other platforms, leaving 113,513 packages with corresponding GitHub links. Finally, after removing 8,784 packages with invalid GitHub links, 104,729 packages remained that correspond to valid GitHub repositories. It is worth noting that in some cases multiple software packages may correspond to the same GitHub repository. In such cases, we do not differentiate between the various development activities, such as commits, pull requests, and issues, that occur within the same GitHub repository across different software packages.

2.1.3 Collecting Activity Data. To collect commit data, we cloned the target Rust repositories to our local platform. Subsequently, we utilized the git log command to capture detailed information for each commit, including SHA, commit author, the author's email, timestamp, etc.

We leveraged the REST API [4] provided by GitHub to retrieve all Pull Requests and Issues for each repository with data field information including the creation time, merge time, closure time, and status of Pull Requests, as well as the creation time, closure time, and status of Issues.

2.2 Data Cleaning

2.2.1 Removing Bots. Bots are widely adopted in various tasks in open source projects [53, 54]. However, bots can distort the true contributions of human developers and the health of the project, thus it is necessary to exclude bots from the analysis. To identify bot accounts, we first removed those that are included in the list of bot accounts compiled by Golzadeh et al. [18]. Then we adopted the method of keyword matching to search bot account using the keyword provided by Schueller et al. [44]. We matched author names with patterns ending in '[bot]', '-bot', or '-bors' [1] , or starting with 'bors', 'dependabot-'. For author emails, after removing the '@' and the domain name that follows, we matched patterns ending in 'bot', 'ghbot', 'bors', 'travis', or '[bot]'. Any matches in either the name or email were considered bot accounts. After these two steps, we manually checked all identified bot accounts. Ultimately, we identified and removed 447 bot accounts.

2.2.2 Merging Developer Identities. It is common for open source developers to use multiple accounts, i.e., a single developer may have multiple names and email addresses [5, 8, 26]. To get an accurate analysis, it is necessary to merge the different identities that belong to the same developer. We addressed this problem by using a rule-based method [59], which augments a developer's name and email address, and has been shown to result in a high level of accuracy (with a precision close to 100%). The merging process reduced the number of developers from 176,035 to 106,791.

2.2.3 Selecting Packages. We followed the following two steps to establish an appropriate sample from the 104,729 packages for further analyses. First, we selected packages with a development duration exceeding one year. Our determination of developer departure is based on the criterion of no commits for over one year (see Sec. 3.1.2), hence we excluded packages released after June 27, 2023. For these packages, it is not possible to determine with certainty whether their developers have left. By applying this criterion, we removed 23,637 packages, leaving 81,092 packages that have been established for more than a year by our dataset's cut-off date of June 27, 2024. Second, we selected packages that have at least one downstream package. As we aim to study the impact of core developer departure on the software supply chain, we excluded 44,101 packages that do not have any downstream packages, leaving 36,991 packages that have at least one downstream package. The dataset that resulted from the steps above was used to answer the four research questions. Sections 3 to 6 each present the analysis procedures and results for the four questions.

3 RQ1: Prevalence of Core Developer Turnover

We measure the prevalence of core developer turnover as follows: (1) how many packages have experienced a loss of core developers; (2) how many packages have lost the core developer who has made the most contributions (which we label the 'top' developer); and (3) how many packages have lost *all* their core developers.

3.1 Method

3.1.1 Identification of Core Developers. Following prior studies [13, 23, 39], we identified the core developers of each package by using a commit-based heuristic, i.e., by considering the number of commits contributed by developers. Commit distributions are heavily skewed in most OSS projects, i.e., a relatively small number of developers make the most contributions, forming a heavy-tailed distribution [39, 58]. Core developers are defined as the top contributors whose commits account for 80% of the total commits in a given project. However, this approach may also include developers

who have made relatively few commits. To address this, we adopted the common practice of using a '5% threshold' to refine our identification of core developers [13, 23]. This means that if a developer's commits represent less than 5% of the total, they are not considered a core developer, even if their commits are included in the top 80%. For example, assuming a situation where the top three developers have respectively contributed 60%, 17%, and 3% of commits (80% in total) in a project, we consider only the first two to be core developers. The first developer, having contributed the most (60%) is called the top developer.

3.1.2 Identification of Developer Turnover. To identify the turnover of core developers, we adopted a one-year threshold [7]. That is, a developer is considered to have abandoned the project if his or her last commit occurred at least one year ago. The cut-off date for our dataset is June 27, 2024. So if a developer's last commit was before June 27, 2023, we considered that the developer has left the project. Additionally, if all developers of a package have left, meaning that the package has not had any commits for over a year, we consider the package to be abandoned.

It can happen that developers still contribute through code reviews or issue handling after ceasing commits. This poses a threat to the accuracy of identifying developer turnover via commit changes. To ensure reliability, we selected a representative sample (374 out of 13,944 repositories) using a 95% confidence level and 5% margin of error [21]. We conducted a manual verification of these repositories, which involved 438 departed core developers. Of those, 377 (86.1%) developers completely disengaged, while 36 temporarily participated in non-coding activities. Only 25 remained active within the year before data collection. Overall, 413 developers (94.3%) showed no activity for over a year, confirming most disengaged completely after stopping code commits.

3.2 Results

We identified the core developers for these 36,991 Rust packages and counted the number of core developers for each package. Figure 1 shows the distribution of the number of core developers. The number of core developers ranges from a minimum of one to a maximum of nine in all the analyzed packages. Noteworthy is the power distribution (note the log scale on the y-axis); by far, most packages (77.4%) have only one core developer. Among those 28,643 packages with one core developer, there are 10,945 (38.2%) packages with only one developer, which includes 7,903 packages (72.2%) that have lost their sole developer, indicating that most packages in the Rust ecosystem are small and sensitive to (core) developer departure and abandonment.

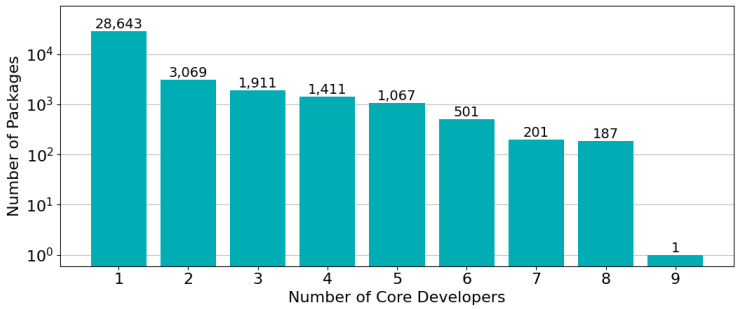


Fig. 1. Distribution of core developer number in the 36,991 Rust packages (note: y-axis uses a log scale)

Considering that most of the packages have only one core developer and are more likely to be poorly maintained or even abandoned after the departure of the single core developer, we divided the sample into two groups: packages with only one core developer (group 1, n=28,643)

and packages with more than one core developer (group 2, n=8,348). Among the 28,643 packages in group 1 that have only one core developer, 16,031 packages were abandoned by their single core developer, accounting for approximately 56% (43.3% of all studied packages). This means that more than half of the packages rely primarily on one developer, and thus face a serious threat to survival.

Table 1. Core developer departure ('Dep') in packages with more than one core developer

Core devel.	No. packages	Core Dep	Top Dep	Dep > 50%	Dep = 100%
2	3,069	1,817 (59.2%)	1,251 (40.8%)	1,817 (59.2%)	968 (31.5%)
3	1,911	1,317 (68.9%)	807 (42.2%)	927 (48.5%)	490 (25.6%)
4	1,411	1,168 (82.8%)	579 (41.0%)	834 (59.1%)	279 (19.8%)
5	1,067	987 (92.5%)	533 (50.0%)	608 (57.0%)	262 (24.6%)
6	501	414 (82.6%)	278 (55.5%)	230 (45.9%)	101 (20.2%)
7	201	161 (80.1%)	71 (35.3%)	154 (76.6%)	53 (26.4%)
8	187	185 (98.9%)	94 (50.3%)	93 (49.7%)	73 (39.0%)
9	1	1 (100%)	1 (100%)	1 (100%)	0 (0%)
All	**8,348**	**6,112 (73.2%)**	**3,614 (43.3%)**	**4,664 (55.9%)**	**2,226 (26.7%)**

We further divided the 8,348 packages in group 2 (i.e., those with more than one core developer) into sub-groups based on the number of core developers, i.e., from two to nine, and counted the different degrees of code developer departure. Specifically, for each sub-group, we calculated the number of packages that faced core developer departure ('Core Dep'), top developer departure ('Top Dep'), losing more than 50% of the core developers (Dep > 50%), and all core developers lost (Dep = 100%), along with their respective proportions (see Table 1). We found that among these 8,348 packages, almost 3 out of 4 (73.2%) have experienced core developer departure, 43.3% have experienced top developer departure, 55.9% have lost more than half of the core developers, and over a quarter (26.7%) have lost all core developers. When compared to packages with one core developer (group 1 above), packages that have more core developers are more prone to losing core developers (larger than 56% in all eight sub-groups, see Table 1). The top developer is usually the creator and maintainer of a package. As shown in Table 1, approximately 43% of packages lost their top developers. Over 20% of packages lost all their core developers. Together, these results highlight a serious threat to the sustainability of the Rust package ecosystem.

Summary for RQ1: Core developer turnover is quite common in the Rust packages. Approximately 77.4% of Rust packages have only one core developer, and 56.0% of those packages have experienced the loss of the core developer, posing a major threat to survival. For the 8,348 packages with more than one core developer: 73.2% of packages have lost core developer(s); 43.3% of packages have lost their top developer; and 26.7% of packages have seen all core developers leave.

4 RQ2: Impact of Core Developer Turnover on Packages They Left

Having established core developer departure as a threat to sustainability, we now investigate the impact on Rust packages. We do so by considering three key development activities in OSS, i.e., submitting code changes, code review, and issue tracking, and comparing these metrics before and after the departure of core developers.

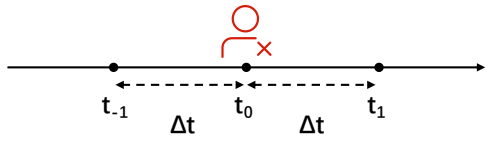


Fig. 2. Example of a core developer departure

4.1 Method

Open source software development involves three key activities, i.e., contributing code changes to repositories, conducting peer review of others' code changes to ensure quality and adherence to project standards, and managing and tracking bugs, feature requests, and other issues [49, 51, 55]. Thus, in this study, we measure impact of core developer departure on these activities. Table 2 presents definitions and metrics.

Core developer turnover can have different impacts on packages with varying numbers of core developers, losing the sole core developer vs. losing one of the core developers in particular. As we observed in relation to RQ1, 7,903 Rust packages only had one developer and lost them, leading to the abandonment of these packages. The impact is quite clear, so we excluded these packages in the analysis for RQ2. We divided the remaining packages into two groups. The first group comprises the 8,128 packages that lost their only core developer, but which had more than one developer. The second group is those packages with more than one core developer and have lost their top developer (n=3,614). Different core developers may leave at different times, so it is difficult to assess the impact of the departure of different core developers on a package. Considering that the top developer is the most important for software development, we take the top developer as the representative of the core developers and focus only on the departure of the top developer. Among these 8,348 packages with more than one core developer, 3,614 packages have experienced the departure of the top developer. We measured the changes in the three activities listed above before and after the only core (top) developer's departure in the two groups, respectively.

If a package A has only one core developer, then t_0 represents the time point at which the sole core developer departs (see Fig. 2); otherwise (with more than one core developer), t_0 indicates the time point at which the top developer depart. t_{-1} and t_1 are the time points Δt before and after t_0, respectively. We assess the impact of the core developer's departure by comparing the differences in the metrics (see Table 2) between time segments $[t_{-1}, t_0]$ and $[t_0, t_1]$.

Code Change Activities. Commits, as the fundamental outputs of project development activities, are directly correlated with the frequency of code changes, the efficiency of software development, and the sustainability of project maintenance [32, 57]. The number of commits within a certain period can reflect the level of development activity and development efficiency of the project

Table 2. Metric for measuring the impact of core developer turnover

Dimension	Metric	Definition
Code change	$\#Commits$	number of commits
Code review	$\#PRs$	number of PRs merged
	$Hours_{PRs,pkg,T}$	median time of all merged PRs for pkg within T
Issue tracking	$\#Issues$	number of issues reported
	$OR_{Issues,pkg,T}$	ratio of issues for pkg within T that remain open at end of T

during that time. We set Δt to 90 days, and compiled statistics on the number of commits (noted as $\#Commits$) between the time intervals $[t_{-1}, t_0]$ and $[t_0, t_1]$ for each package.

Code Review Activities. Pull Requests (PRs) are formal requests for code change review, which help in delivering software quality and allow transparency of the development process [28]. Similar to commits, the number of PRs merged within a certain time frame of a project can reflect development activity. Considering that PRs typically involve a code review process, they may require more time to complete and have a longer activity cycle [25], we set Δt to 180 days, and compiled statistics on the number of PRs ($\#PRs$) merged between time intervals $[t_{-1}, t_0]$ and $[t_0, t_1]$.

Further, we also consider the time required for processing PRs before and after the departure of a core developer to measure code review efficiency. We note $Hours_{PR}$ to represent the hours spent on processing a PR, which is obtained by subtracting the creation time from the merge time of the PR. For each package and time interval T, we calculate the median of $Hours_{PR}$ of all PRs merged in T to represent each package's overall level of time spent on processing PRs in time interval T, noted as $Hours_{PRs,pkg,T}$. Similarly, we set Δt to 180 days and calculated the $Hours_{PRs,pkg,T}$ for each package within the time intervals $[t_{-1}, t_0]$ and $[t_0, t_1]$.

Issue Tracking Activity. Issue tracking activity in OSS projects refers to reporting bugs, requesting features, starting discussions, solving issues, and so on [50]. The number of reported issues in a certain period can reflect the level of user attention to the project as well as the degree of community involvement in the project [9]. Similar to PRs, the activity cycle for issues is also long. Therefore, we set Δt to 180 days, and compiled statistics on the number of issues (noted as $\#Issues$) reported between the time intervals $[t_{-1}, t_0]$ and $[t_0, t_1]$.

Further, the proportion of unsolved reported issues can reflect the feedback efficiency and maintenance ability of OSS projects. Thus, we calculate the ratio of issues for package pkg that were reported within the time intervals $[t_{-1}, t_0]$ and $[t_0, t_1]$ and remained open at the end of these respective time intervals, t_0 and t_1. We denote this ratio as $OR_{Issues,pkg,T}$, where T represents $[t_{-1}, t_0]$ and $[t_0, t_1]$ respectively. The larger the value of $OR_{Issues,pkg,T}$, the lower the issue resolution efficiency for pkg within T.

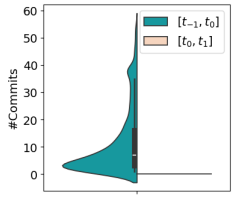
Statistical Analysis. To assess whether there are significant differences between two paired data sets, we employed the Wilcoxon signed-rank test [41]. This non-parametric statistical method is suitable for analyzing paired sample data when the assumptions of normal distribution or homoscedasticity are not met. The null hypothesis states there is no difference between the paired groups. Given that we conduct approximately ten independent Wilcoxon signed-rank tests, to control the overall error rate (Family-Wise Error Rate, FWER) and to mitigate the risk of false positives due to multiple comparisons, we applied the Bonferroni correction method [46]. By dividing the traditional significance level $\alpha = 0.05$ by the number of tests, we have adjusted the significance threshold for each test to 0.005. Upon completion of the tests, if the p-value is less than or equal to the adjusted significance level, we reject the null hypothesis and conclude that there is a statistically significant difference between the paired groups.

The magnitude of this difference is quantified using Cliff's Delta d, which provides a standardized measure of effect size. Cliff's Delta [30] has an absolute value ranging from 0 to 1, where a larger value indicates a more pronounced effect. To interpret the value of d, we adopt the thresholds proposed by Romano et al. [40].[1]

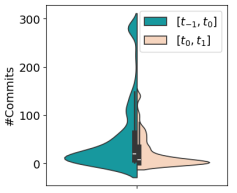
4.2 Result

4.2.1 Impact on Code Change Activities. Figure 3 shows the change in the number of commits for two groups of packages, 90 days before and after the departure of the core (top) developer.

[1]negligible ($|d| \leq 0.147$), small ($0.147 < |d| \leq 0.33$), medium ($0.33 < |d| \leq 0.474$), or large ($0.474 < |d| \leq 1$)



(a) Packages with only one core developer



(b) Packages with more than one core developer

Fig. 3. Number of commits before (left side of violin plot) and after (right side of violin plot) core developer depart, for packages with one core developer (a) and packages with more than one core developer (b)

For the 8,128 packages with only one core developer (Fig. 3a), the number of commits significantly decreased after the sole core developer left, with the median dropping from 8 to 0 and the maximum value decreasing from 1,442 to 1,135. Specifically, an overwhelming 85.2% of these packages eventually ceased to be maintained after the departure of the sole core developer. More significantly, 63.4% of the 8,128 packages ceased maintenance immediately upon the departure of the core developer. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta result was large (d = -0.84).

For the 3,614 packages with more than one core developer (Fig. 3b), there was a decrease in the number of commits after the departure of the top developer, with the median dropping from 29 to 13. Among them, 48.8% of packages saw maintenance activities cease, and 11.6% of packages ceased maintenance immediately upon the departure of the top developer. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta result was small (d = -0.27).

We found that the departure of a core (top) developer is associated with a decrease in the number of commits for packages, whether they have only one core developer or more than one. However, packages with more than one core developer experienced a less severe reduction, indicating that such projects demonstrate a stronger resistance to risk as other core developers are more likely to continue the project's maintenance after the departure of the top developer.

4.2.2 Impact on Code Review Activities. Figure 4 shows the distributions of merged PR numbers and PR processing hours within 180 days before and after the departure of the only or top core developer in the two package groups, i.e., 8,128 packages with only one core developer and 3,614 packages with multiple core developers, respectively.

For the 8,128 packages in the first group (see Figure 4a), the number of PRs significantly decreased after the sole core developer left, with the median dropping from 1 to 0 and the maximum value decreasing from 302 to 242. More than half of the packages no longer had PRs merged after the departure of their only core developer. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta was medium (d = -0.38). Further, we focused on the changes in the hours required to merge PRs for these packages. Out of the 8,128 packages, 1,758 packages had PRs both during the 180 days before and after the core developer's departure. We calculated the $Hours_{PRs,pkg,T}$ for these packages and found that after the core developer left, the $Hours_{PRs,pkg,T}$ for these packages generally increased, with the median rising from 5.5 hours to 15.4 hours. A Wilcoxon signed-rank test indicated a significant difference (p-value of .00037 (< .005)). The Cliff's Delta result was negligible (d = 0.09). This suggests that after the sole core developer's departure, the project's development activity decreased, and the review process slowed down.

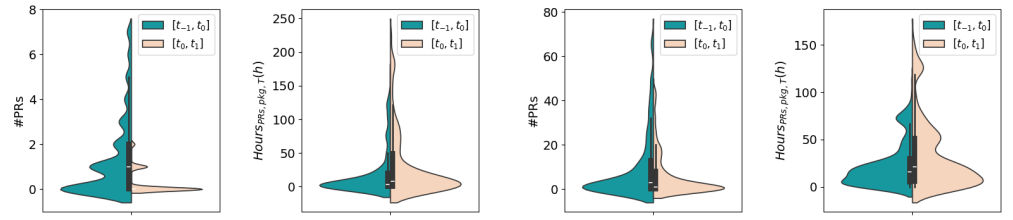
In the second group, there are 3,614 packages with multiple core developers that have experienced the departure of top developers. As shown in Figure 4b, for these packages, the number of PRs has

decreased after the top developers left, with the median dropping from 5 to 3. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta result was negligible (d = −0.1). Among these 3,614 packages, 2,100 packages had merged PRs both during the 180 days before and after the departure of the top developers. Calculating the $Hours_{PRs,pkg,T}$ for each package, we found that after the top developers left, the median $Hours_{PRs,pkg,T}$ increased slightly, from 20.3 to 23.9. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta result was negligible (d = 0.09). This indicates that packages with multiple core developers will also experience a decline in community activeness after the departure of top developers, but compared to packages with only one core developer, the impact is much less significant. Unsurprisingly, this suggests that the presence of other core developers helps such projects maintain community participation, even after losing core developers.

4.2.3 Impact on Issue Tracking Activities. Figure 5 shows the distributions of number of reported issues and the ratio of these issues that remain open at the end of $[t_{-1}, t_0]$ and $[t_0, t_1]$ within 180 days before and after the departure of the only or top core developer in the two groups, i.e., 8,128 packages with one core developer and 3,614 packages with multiple core developers, respectively.

For group 1 (8,128 packages, see Figure 5a), the number of reported issues decreased after the sole core developer left, with the maximum value decreasing from 163 to 49. Notably, packages with only one core developer generally had few issues, with the median number of issues remaining 0 before and after the developer's departure. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta result was small (d = −0.15). Out of 8,128 packages, 1,804 packages had reported issues both during the 180 days before and after the sole core developer's departure. We calculated the $OR_{issues,pkg,T}$ for these packages. After the sole core developer left, the $OR_{issues,pkg,T}$ generally increased, with the median rising from 0.57 to 1, which means that more than half of the packages have practically lost their issue resolution capability after the departure of the sole core developer. A Wilcoxon signed-rank test indicated a significant difference (p-value ≈ 0). The Cliff's Delta result was small (d = 0.3). This suggests that after the sore core developer's departure, the project's user attention and community engagement have decreased, along with a reduction in issue resolution efficacy.

For group 2 (3,614 packages, see Figure 5b), we found that the number of reported issues showed no significant change, or even a slight increase after the departure of the top developer. Specifically, the median number of issues remains 2 before and after the developer's departure. The Wilcoxon signed-rank test showed no significant difference (p-value = 0.219). Of 3,614 packages, 1,900 packages had issues both during the 180 days before and after the top developer's departure. We



(a) Packages with only one core developer (b) Packages with more than one core developer

Fig. 4. *#PRs* and *Hours* before (left side of violin plot) and after (right side of violin plot) core developer departed, for packages with one core developer (a) and packages with more than one core developer (b)

calculated the $OR_{issues,pkg,T}$ for these packages and observed an increase after the departure of the top developer, with the median value rising from 0.5 to 0.54. A Wilcoxon signed-rank test yielded a significant difference (p-value < .005). The Cliff's Delta result was negligible (d = 0.06). This suggests that following the departure of the top developer, packages with multiple core developers did not experience a significant change in the number of issues, and still maintained a high level of user attention and community engagement. However, the efficiency of issue resolution noticeably decreased, but due to the presence of other core developers, this decrease is less in comparison to packages that had only one core developer.

Summary 1 for RQ2: *For packages with only one core developer:* We found that the departure of core developers significantly affects these packages, i.e., the number of commits decreases markedly, leading to reduced development and maintenance efficiency; approximately 85.22% of packages cease to be maintained; the number of pull requests and issues declines, while the time to process pull requests and the ratio of unresolved issues increase.
Summary 2 for RQ2: *For packages with multiple core developers:* The departure of the top developer leads to a decrease in commits and merged PRs, a slight rise in PR processing time, and an increase in the ratio of unresolved issues. However, the presence of multiple core developers mitigates these impacts (only 48.8% of the packages cease to be maintained), demonstrating a stronger resilience to risks.

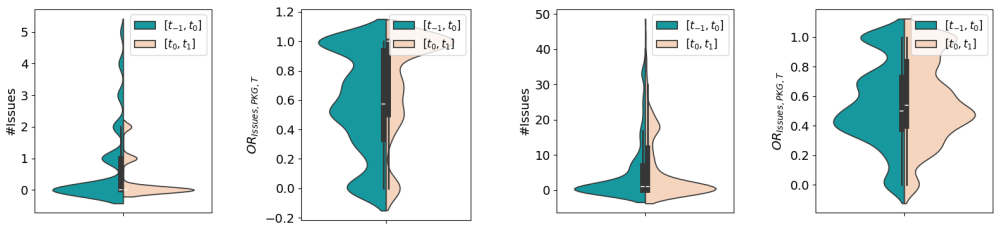
5 RQ3: Downstream Affected by Upstream Core Developer Turnover

We now consider the impact of core developer departure on downstream packages (RQ3). We answer this RQ by quantifying the number of packages that are directly or indirectly dependent on packages losing core developers.

5.1 Method

Direct and indirect dependencies. We retrieved the direct dependency relationships between all packages in the Rust ecosystem from the database export of *crates.io*, compiled a list of direct dependencies for each package, and obtained a list of all dependencies for each package (including both direct and indirect dependencies) using a breadth-first search method.

Different types of core developer turnover. The impact of different types of core developer turnover on downstream projects varies. In this study, we classified packages with departed core developers into three categories: (1) losing the *top* developer; (2) losing *all* core developers; and (3) ceased *all* maintenance. Out of 36,991 packages, 19,645 packages exhibit *top* developer



(a) Packages with only one core developer (b) Packages with more than one core developer

Fig. 5. *#Issues* and *OR* before (left side of violin plot) and after (right side of violin plot) core developer departed, for packages with one core developer (a) and packages with more than one core developer (b)

departure, 18,257 packages have *complete* core developer departure, and 16,593 packages have ceased maintenance.

Quantify upstream dependencies affected by core developer turnover. For each package P, we calculated the proportion of its direct dependencies with three different types of core developer departure, which are defined as RDT (Ratio of Direct Dependencies having Top Developer Departure), RDC (Ratio of Direct Dependencies having All Core Developer Departure), and RDM (Ratio of Direct Dependencies having Ceased Maintenance). Eqn. (1) shows how we calculated the three ratios. Among them, $\#dirdep$ represents the number of direct dependencies for package P, $\#dirdep_{\text{topdepart}}$, $\#dirdep_{\text{coredepart}}$, and $\#dirdep_{\text{ceasedmaint}}$ represent the number of package P's direct dependencies with top developer departure, all core developer departure, and ceased maintenance, respectively. Similarly, for each package P, we also calculated the proportion of all its direct and indirect dependencies with the three types of core developer departure, as shown in eqn. (2).

$$RDT = \frac{\#dirdep_{topdepart}}{\#dirdep} \qquad RDC = \frac{\#dirdep_{coredepart}}{\#dirdep} \qquad RDM = \frac{\#dirdep_{ceasedmaint}}{\#dirdep} \tag{1}$$

$$RAT = \frac{\#alldep_{topdepart}}{\#alldep} \qquad RAC = \frac{\#alldep_{coredepart}}{\#alldep} \qquad RAM = \frac{\#alldep_{ceasedmaint}}{\#alldep} \tag{2}$$

5.2 Results

We first counted the number of direct dependencies for each package in the Rust ecosystem. Among these 149,445 packages, 36,126 packages have *no* dependencies. Across all packages, the median number of direct dependencies is 3, while the highest number we found is 161. For the 113,319 packages having dependencies, we calculated the ratio of core developer departure for three different types within the direct dependencies of each package (the left halves of the double-sided violin plots in Figure 6). These packages are very commonly directly affected by the departure of top developers from upstream dependencies. 61.2% of packages have at least one direct dependency on a package that has experienced top developer departure. The median RDT of these packages reached 0.2, indicating that over half of the packages have seen top developer turnover in at least one-fifth of their direct dependencies. Additionally, close to half (45.9%) of the packages have at least one direct dependency that has lost all of its core developers. Developers who use these packages that have lost all core developers may find it difficult to receive timely support when encountering issues. More significantly, more than one-third (35.7%) of packages depend on packages that have ceased maintenance, which poses a significant threat to the security of such packages.

Considering that indirect dependencies may also affect packages through the software supply chain, we calculated the proportion of packages with core developer departure across three different types among all dependencies of these packages. We counted the number of dependencies, both direct and indirect, for these 149,445 Rust packages and found that the median number of dependencies is 10, with the maximum being 778. The right halves of the double-sided violin plots in Figure 6 illustrate the proportion of packages with developer departure for three different types among all dependencies for each of the 113,319 packages. When taking into account indirect effects, the impact on downstream is even more severe. The proportion of packages having upstream with three types of core developer departure reached astonishing levels at 77.0%, 69.1%, and 64.1% respectively. These results indicate that the vast majority of packages in the Rust ecosystem are at risk of being directly or indirectly affected by the loss of upstream core developers.

Summary for RQ3: The departure of core developers not only affects their maintenance and sustainability but also directly or indirectly impacts downstream software packages in the ecosystem. Among 113,319 Rust packages with upstream dependencies, 61.2%, 45.9%, and 35.7% of packages are directly affected by their upstream top developers' departure, complete loss of core developers, and ceased maintenance, respectively. More significantly, taking into account the effects of indirect dependencies on the software packages, the respective percentages increase to 77.0%, 69.1%, and 64.1%.

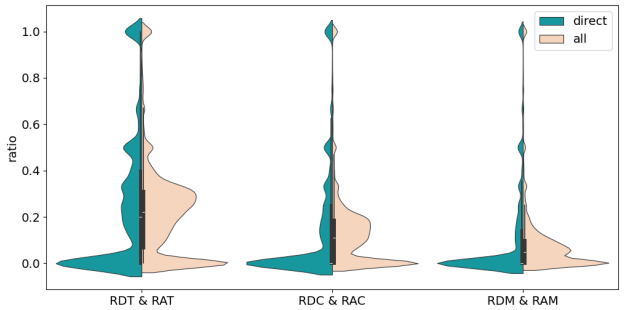


Fig. 6. Impact of core developer turnover on 113,319 downstream packages. The left halves of the double-sided violin plots represent ratios of three different types of core developer loss in the direct dependencies: RDT (top developer departure), RDC (losing all core developers), and RDM (ceased maintenance). The right halves of the double-sided violin plots represent ratios of three different types of core developer loss in all dependencies: RAT (top developer departure), RAC (losing all core developers), and RAM (ceased maintenance). More than 60% of packages have at least one direct dependency with top developer departure; this risk amplifies to 77% when indirect dependencies are included.

6 RQ4: Perspectives on Upstream Core Developer Turnover

Finally, we conducted a survey among downstream package maintainers who rely on upstream packages that have faced core developer departure (RQ4).

6.1 Method

To understand the perspectives of downstream developers on the departure of core developers from upstream dependencies, we surveyed maintainers in downstream packages. We first selected packages affected by the departure of core developers from upstream dependencies (packages with core developer departure in their direct dependencies), then selected those maintainers whose public emails were available on GitHub. Additionally, we selected the top 510 most active packages based on the number of commits in the past year and sent out a survey invitation to their maintainers.

In the survey, we first explored whether developers have recognized the key role of core developers in OSS projects by asking *"How important do you think core developers are for the sustainable development of open source projects?"*, with a 5-point Likert scale [22] (5 = Very important, 1 = Very unimportant). Following this, we explored how they dealt with upstream core developer departure when relying on upstream. Considering the complexity and long time cost in detecting the upstream core developer departure, we also asked their views on receiving a real-time notification, intending to guide future work. Detailed questions can be found in our online survey [6].

We sent 510 invitations (11 failed to be delivered). The survey remained open for two months; we received 40 responses, with a response rate as $\frac{40}{510-11} = 8.02\%$. To protect the privacy of the

respondents, we label them as $P_1 - P_{40}$ respectively. The 40 respondents have a good representation: they come from 22 different nationalities; across male, female, and non-binary genders; and most of them have more than one year of Rust experience and have been involved in 3-5 Rust projects.

In our analysis of the open-ended survey questions, we used thematic analysis [45]. This involved carefully reading through the responses to understand the main ideas. The first two authors independently coded key points of the answers and grouped similar codes into themes, and conflicts were resolved by several face-to-face meetings. Through this process, we identified the main concerns and opinions of the participants regarding the impact of core developer departures in upstream packages. Details of the survey and the process documents of the thematic analysis (including the codes of responses, the generated themes, and the consolidated themes) are available in the online appendix [6].

6.2 Results

6.2.1 Perspectives on the Importance of Core Developers. We first asked the respondents to rate the importance of core developers and inquired about their perspectives. Of the 40 respondents, 17 selected 'Very important', 16 selected 'Important.' 4 held a neutral stand, and 3 deemed 'Unimportant.'

Approximately 82.5% of respondents believe that core developers are crucial to OSS projects, especially during the initial development phase of the project. They believe core developers have an in-depth understanding of the project goals, direction, and codebase, and can drive the project forward in a leadership position. Further, they deem core developers to be responsible for feature expansion, code quality assurance, and addressing issues, and play a key role in OSS project development and maintenance. For example, one respondent (P_{20}) wrote: *"I believe it is very important. Core developers determine the main programming paradigms, code quality, scalability, and direction of development for the project."*

The three respondents who chose the 'Unimportant' option held the view that OSS *communities* are more critical because they can provide ongoing contributions and maintenance, keeping the project vibrant even after the departure of core developers. For example, P_3 shared: *"it's not that much important, what is important is the community support for it, for example, the main developer of actix web left, but the library is still widely used and maintained."*

6.2.2 Response to Upstream Core Developer Turnover. We surveyed developers on whether they would consider the departure of core developers when choosing a dependency, whether they can recognize the departure of core developers from certain signs, and the possible measures they will take when the departure of upstream core developers occurs.We also asked for their reasons.

Sixteen (out of 40, 40%) respondents indicated that they would consider core developer departure when selecting an upstream package to use, while 18 respondents chose "Maybe," and only six developers chose "No." Reasons for considering this factor are mostly the same as their views on the importance of core developers on OSS projects, i.e., relying on core developers to actively maintain a package and update it promptly. For respondents who chose the "Maybe" option, we found their uncertainties mainly occurred when considering whether the package (1) is small-scale and functionally stabilized; (2) still works and bugs can be fixed by non-core developers or themselves; (3) is still maintained by an active community with multiple core developers. For example, P_{37} said: *"It depends on if the package is simple or solves a problem specific enough to be considered "finished" and without bugs. In that case, I would not mind [adding it as a dependency]."*

We then asked developers: *"What specific signs would make you feel like core developers are leaving the upstream software packages?"* One developer can provide multiple signs. Thirty-two respondents indicated that they make their judgments when "issues submitted have not received a response for a long time"; 30 developers mentioned the sign that "pull Requests have not been merged for a

long time"; and 27 individuals also opted for the sign "The software package has not been updated for a long time". The three signs show a commonness, i.e., slow or even no community feedback or software updates. These signs are often only perceived after the core developers have left for a while. Downstream projects may be exposed to risks because of possible upstream unfixed vulnerabilities and miss an opportunity to retain core developers. Further, four respondents indicated that they were "not aware of the departure of core developers of the packages." A lack of such awareness may therefore misinform developers' decision as to whether to use a package.

We asked developers to imagine a scenario where core contributors, especially the primary developer, left the upstream packages they depended on; we explored how they would respond. We found that developers often do not take action immediately. Thirty-one individuals chose to "Keep using it until serious problems happen." Twenty-seven developers said they would "Contribute to the upstream when necessary." On the contrary, 24 respondents would "Migrate to other appropriate packages." A small number of respondents (n=4) indicated that they could "Donate to the upstream package" to retain the core developers.

If the departure of core developers has't led to serious consequences, developers may not be aware of any risks, and only take measures when issues arise in the project. In the face of upstream core developers departing, developers may make different choices depending on the situation.

6.2.3 Real-Time Notification of Upstream Core Developer Turnover. Given the importance of core developers to OSS projects and the lack of a real-time mechanism to become aware of their departure upstream, we asked whether developers would be willing to receive real-time notifications and reasons. If their answer was "Yes" or "Maybe," we also invited them to share what kinds of information they would like such notification to include.

Eleven respondents chose "Yes," 20 chose "Maybe," and only 9 chose "No." Besides simply stating "good/important/desirable to know", reasons for the 11 respondents indicating 'yes' included: (1) preparing for replacement; (2) being instrumental when choosing a project to use; (3) handing over maintenance immediately; and (4) overcoming the limitations of inactive monitors and RustSec's unmaintained advisory.[2] For example, P_{17} said: *"I am not sure how this software will quantify a 'departure.' But if it is accurate, it would be very instrumental when choosing a project to use."*

Those who responded "Maybe" and "No" raised three reasons: (1) the upstream project may be still stable and work well; (2) notifications may add pressure on package owners; and (3) notifications may cause unnecessary concern or panic. Thus, it appears that developers have different tolerance levels for dependency risks. Approximately 17.2% of respondents indicated that if their upstream does not have security and stability issues, they would keep using it even when upstream core developers leave. For instance, P_{15} said: *"As long as the current version of the package works for our use case, there is no need to do anything about it."* On the other hand, the other two concerns are worth addressing when designing a departure notification, e.g., improving the identification accuracy of core developers' departure, forecasting possible impacts, and exploring a soft but efficient way to notify downstream.

In terms of what kind of notification information they would like to receive, we set up a multiple-choice question with an open-ended option. Twenty-six respondents chose "Recommendations for alternative software packages"; 21 people selected "Real-time notifications of the maintenance status of affected software packages"; 20 individuals opted for "Real-time notifications of core developer departure"; and 17 individuals chose "Reasons for core developer departure." Compared to the departure of core developers and the reasons behind it, developers are more concerned with the short-term impact on project maintenance and available alternatives in case the upstream package

[2]RustSec is the Rust Security Advisory Database, which helps developers identify and fix security vulnerabilities in Rust packages.

ceases to be maintained. No respondent provided new information that should be mentioned in the notification.

Summary for RQ4: Respondents generally understood that the departure of core developers may impact the stability and direction of the project, but a few of them think an active and supportive community can mitigate these effects and maintain the project's ongoing development. In addition, downstream developers place a high value on transparency and timely notifications regarding the health status of upstream dependencies, so that they can make risk management decisions.

7 Implications

We discuss a number of implications resulting from the findings of this study, for several different stakeholders. We discuss them in turn.

7.1 Implications for Package Owners

Enhance community participation and develop a succession plan. Active community involvement, as emphasized by survey respondents, is vital for a project's sustainability, especially in the event of a core developer's departure. Package owners should foster an environment that encourages diverse contributions, ensures effective communication, and provides clear documentation and guidelines. To further boost community participation, package owners can introduce a contributor classification system (such as "core developers - active contributors - newcomers"). This system allows for better recognition of different levels of contributions and taking timely and specific measures to retain them. Besides, different incentives can be provided for contributors at various levels. Package owners should measure the activity of different contributors and provide rewards when necessary to alleviate their economic crisis, core developers in particular. Projects with multiple core developers are more resilient in the face of developer departures (Section 4). By creating a succession plan and gradually transferring responsibilities to trained community members, package owners can ensure leadership is shared among core developers, increasing the project's resilience.

Declare maintenance status. Package owners should declare maintenance status via means like the *README* file when core developers depart, giving users time to handle potential risks.

Automate the maintenance process. Package owners should integrate automated tools such as GitHub Actions to automate key maintenance tasks. For instance, they can use Dependabot for dependency updates, Rust CI for basic testing, and rustdoc for documentation generation. By automating repetitive tasks, this approach reduces reliance on core developers and may ensure the project continues smoothly even when they are unavailable.

7.2 Implications for Downstream Users

Enhance monitoring of dependency health. The departure of core developers is often a reason for a package to cease maintenance and also a harbinger of such a stoppage. Downstream users should take into account indicators from upstream such as code update frequency, speed of issue resolution, and community activity levels. If there is a loss of core developers upstream, they should promptly assess the risks and develop strategies to address them, such as seeking alternative packages, to ensure the stability and security of their projects.

Keep an eye on possible alternatives to key dependencies. Given the high turnover of core developers in the Rust ecosystem, downstream users should regularly research and assess alternative libraries or frameworks that offer similar functionality, in case the main dependencies face issues. Maintaining

flexibility and modularity in the software architecture can ease the replacement of dependencies without requiring major refactoring.

7.3 Implications for the Open Source Community

Developer departure early-warning system. Design a prediction model based on contribution activities. The dimensions identified in our study, such as developer commit frequency and issue response delay, can be used as features. A classifier can be trained using historical data to identify potential departure risks. The community can make such information publicly available through a "health dashboard" for downstream users' reference. Additionally, OSS communities should also provide a range of support measures, including recommending alternative libraries or tools and offering guidance for migration.

7.4 Implications for Researchers

Need further investigation on the existence of projects that do not require ongoing maintenance and what their characteristics are. In our survey, several participants mentioned that if a project is mature, it will not encounter problems even if core developers leave. Surprisingly, we found that only 40% of respondents indicated that they would consider core developer departure when selecting an upstream package to use. Are there some projects that do not require ongoing maintenance because of certain characteristics they possess? For instance, small-size projects do not have a lot of code to maintain. When their core developers depart, someone can step in and understand the issue easily. This size factor, along with the importance of the project, like the number of dependencies, can serve as a proxy for how much a project needs to be maintained and can help quantify the impact of developers' departures. Other types of projects that are considered to not need continuous maintenance include complete projects, stable projects that have not changed for a long time, or simple projects. Researchers can build on this to further investigate how these maintenance-free or self-sustaining projects differ in resilience and dependency risks from those that rely on constant updates. One potential avenue for this is to link this to Lehman's SPE classification [27], whereby S-type systems need less maintenance [cf. 11].

Further explore the impact of core developer turnover on downstream. This study focuses on core developer turnover in Rust's software supply chain ecosystem. In RQ3, we evaluated the overall impact on the downstream in the Rust ecosystem by counting the number of downstream packages affected by the turnover of upstream core developers. Researchers can conduct specific studies, based on our research, on the impact of the turnover of upstream core developers on downstream functionality, security, and project performance within small-scale samples in the future.

8 Threats to Validity

We are aware of several threats to validity which we discuss next.

Internal validity: Rust package repositories could be hosted on various platforms, such as GitHub and GitLab. To keep the explored objects hosted on the same platform, we focused on packages with repositories on GitHub. Because the vast majority (93.8%) of package repositories are located on GitHub. We deem studying Rust packages on GitHub can gain a representative understanding of the prevalence, impact, and developers' perspectives of core developer departure.

A second threat lies in determining developer departure, i.e., if a developer had not made any commits for over a year, we considered them as left. Accurately identifying whether a developer has truly left an OSS project is complex. One prior study [7] conducted a sensitivity analysis by comparing different thresholds to determine developer turnover, i.e., 3 months, 6 months, 1 year, 1.5 years, and 2 years. They found that the one-year threshold was the least sensitive to error while accurately identifying developers who had truly left the project. Scholtes et al. [43] find that 90% of

consecutive commits occurred within periods less than 295 days, which implies that if a developer does not make a commit for over 295 days, the likelihood of them committing again in the future is less than 10%. Consequently, if a developer goes beyond 365 days without making a commit, the probability of them making a commit again would be extremely small. Thus, we deem the method we adopted is appropriate.

For RQ4, we collected publicly available developer emails on GitHub for our survey. To ease ethical concerns, especially GitHub's Acceptable Use Policy [3] we took several steps. First, to avoid disrupting too many users, we chose 510 active individuals who are the owners of the libraries and who deliberately disclosed their email addresses on their GitHub homepages. Since GitHub users' email addresses aren't shown by default, this likely indicates their willingness to participate in OSS-related activities, including research-oriented surveys. Second, our emails, besides being survey invitations, also reminded library owners about the turnover of their upstreams' core developers and associated risks. We considered this a helpful reminder, distinct from the 'spamming purposes' prohibited by GitHub's policy. In our emails, we also informed recipients of the research aim of this study and the anonymous nature of responding. The survey passed the review by the ethics review board at our university. Finally, we did not receive any 'negative' responses such as "do not contact"; in fact, several respondents expressed their support for this research. For example, P_{20} said: *"That's all for now. If you need further advice, please feel free to email me. I'm happy to provide assistance."* Besides, the survey questions regarding core developers' importance may tend to receive positive answers because of open source nature and inherent dependency relationship. As shown by the survey results, nearly 60% of respondents consider core developer departure unimportant when selecting upstream packages. The percentage is not low, indicating that the question framing risks have a limited impact on developers' feedback.

External validity: Our study mainly focuses on conducting a detailed examination of the Rust ecosystem. As Table 3 shows, several studies have investigated the prevalence and impact of core developer turnover in some specific OSS projects. However, none of these studies have considered the impact of developer turnover at the level of a programming language (PL) ecosystem. To bridge the knowledge gap, we selected one popular and emergent ecosystem, i.e., Rust with 149,445 software packages and 106,791 developers, to conduct an in-depth exploration first. Our study makes novel contributions to the literature on developer turnover. In addition, many programming languages have commonalities, such as package hosting platforms and reliance on active communities. Thus, our findings regarding core developer turnover in the Rust ecosystem may be extended to other languages. For other PL ecosystems, the differences in the prevalence and impact of core developer turnover are a worthy research task. Future researchers can build on our measurement framework to study core developer turnover in other language ecosystems.

9 Related Work

Prior work has revealed a variety of reasons why core developers might leave a project [19, 20, 31]. For instance, Miller et al. [31] found that career transitions are a common factor leading to developer disengagement. Gray [19] emphasizes the impact of community hostility, lack of support, and changes in project direction on developer disengagement. Furthermore, Iaffaldano et al. [20] found that personal life events and project-related factors (such as community dynamics and governance issues) also prompt developers to pause or cease their contributions. These studies collectively demonstrate that developer disengagement from OSS projects is a complex phenomenon, influenced by a variety of factors at the individual, community, and project levels.

A key focus in this paper is to determine the impact of core developer departure in OSS projects. Prior work has focused on several specific OSS projects [7, 16, 17, 36, 37, 42] to study the characteristics and impact of core developer turnover. Such impacts include knowledge loss and other negative

effects [15, 33–36, 38], which in turn affect the maintenance and development of the project. Table 3 presents a summary of several key studies in this area. Rigby et al. [36] investigated how to mitigate knowledge loss by predicting successors and employing risk management measures. Robillard et al. [38] uncovered the various contexts of knowledge loss in practice and its impact on software projects. Additionally, some work studied patterns of developer turnover in OSS projects, project survival rates, and the impact on productivity. Foucault et al. [17] explored the impact of developer turnover on software quality and found a significant negative correlation between the activity of external newcomers and software quality issues. On the other hand, Avelino et al. [7] analyzed the abandonment and survival of OSS projects following the loss of key developers, exploring the survival rates, differences between surviving and non-surviving projects, and the motivations and challenges faced by new core developers who take over the projects. Lastly, Russo Latona et al. [42] employed a difference-in-differences analysis to quantify the impact of core developer dropout on the productivity of 9,573 OSS projects on GitHub, revealing a significant decline in productivity and underscoring the need for resilience in OSS project management.

This study extends this literature in three ways. First, we study core developer departure and its impact at the level of a whole ecosystem, rather than an individual project. Second, prior studies typically measured the impact of core developer turnover from isolated aspects, for example, by measuring the density of bug fixes [17], the number of abandoned files [36], and the issue resolution time [16]. In contrast, this study provides a more comprehensive analysis across three interconnected activity dimensions (i.e., code change, code review, and issue tracking). Third, prior studies mainly focused on the impact of core developer turnover within the projects they left. This study is the first to examine the effect on the software supply chain by studying the impact on downstream projects, and we include an analysis of downstream package owners' perceptions of upstream core developer turnover.

10 Conclusion

Core developer turnover can be catastrophic for an OSS project. Modern software development highly relies on the use of open source libraries, but very little is currently known about the core developer turnover in the context of such software ecosystems, which represent software dependency networks. Whereas prior work has investigated developer turnover within the context of a single project, this paper is the first study to comprehensively examine developer turnover in an OSS ecosystem. We find that core developer departure is prevalent in the Rust package ecosystem: 77.4% of the 36,991 packages have only one core developer, and 56.0% of these packages have lost their sole developer; among the remaining 8348 packages with multiple core developers, 73.2% have experienced core developer departure, 43.3% have lost their top developer, and 26.7% of packages have seen all core developers leave. We measured the impact of core developer departure on three key development activities and found a noticeable decrease in code contribution frequency and a significantly longer time to process code reviews and fix issues after the departure of core developers. Besides, the effects of core developer departure extend to downstream dependencies, since 61.2%, 45.9%, and 35.7% of packages have upstream dependencies that undergo the departure of the top developer, complete loss of core developers, and cessation of library maintenance, respectively. In the last, we explore developers' perspectives toward core developer departure in the context of software dependency construction. We find that most respondents acknowledge the importance of core developers and are willing to accept a timely notification of their upstream dependencies' core developer status. The results of our study demonstrate the necessity of considering core developer turnover when building software dependencies.

Table 3. Comparison with related work on developer turnover

Study	Sample	Findings
Foucault et al. (2015) [17]	Five large open-source projects	• Turnover is common in OSS projects, with at least 80% of developers being new or leaving. • There is a positive correlation between the external newcomer activity and the density of bug fixes. indicating a negative impact on software quality.
Rigby et al. (2016) [36]	Chrome and Avaya	• The actual losses may be more severe than historical losses, and although the extreme "truck factor" scenarios have large losses, the probability is low. • Recommending potential successors based on file co-change relationships can reduce the risk caused by developer turnover.
Avelino et al. (2019) [7]	1,932 popular GitHub projects	• 16% of the projects experienced Truck Factor Developers Detachment (TFDD), and 41% of the projects survived after TFDD. • New TF developers were often motivated to participate because of their use of the project, and human and social factors played a key role in attracting new TF developers.
Ferreira et al. (2020) [16]	174 FLOSS projects	• Core developer turnover is common in FLOSS projects, with significant annual rates, affected by ownership, team size, and language. • Projects vary in stability based on core developer inflow and outflow; unstable ones take much longer to address issues, bugs, and enhancements than others.
Robillard et al. (2021) [37]	three software companies	• Staff turnover in software projects causes complex knowledge loss, covering permanent/temporary, sudden/expected, and complete/partial departures. • Knowledge loss impacts include lack of guidance, reliance on docs, colleague collaboration, and knowledge reconstruction, all affecting software dev quality and efficiency.
Russo Latona et al. (2024) [42]	9,573 OSS GitHub projects	• The dropout of core developers leads to a decrease in project productivity, with the productivity of the remaining developers dropping by up to 20%, and this effect lasts for at least 16 weeks. • Projects with higher productivity before the dropout of core developers are more vulnerable to shocks, indicating a trade-off between productivity and resilience.

Data Availability

To facilitate future study of core developer turnover in package ecosystems of other programming languages, our data and analysis scripts are publicly available in a replication package [6].

Acknowledgments

This work is supported by the National Natural Science Foundation of China Grants (62202048, 62232003, and 62141209), Taighde Éireann – Research Ireland under Grant number 13/RC/2094_P2, and CCF-Huawei Populus Grove Fund.

References

[1] [n. d.]. bors.tech. https://bors.tech/.

[2] [n. d.]. crates.io: The Rust community's crate registry. https://crates.io.

[3] [n. d.]. GitHub Acceptable Use Policies - Information Usage Restrictions. https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies#7-information-usage-restrictions.

[4] [n. d.]. GitHub REST API documentation. https://docs.github.com/en/rest.

[5] Sadika Amreen, Audris Mockus, Russell Zaretzki, Christopher Bogart, and Yuxia Zhang. 2020. ALFAA: Active Learning Fingerprint based Anti-Aliasing for correcting developer identity errors in version control systems. *Empirical Software Engineering* 25 (2020), 1136–1167.

[6] Anonymous. [n. d.]. Online Appendix. https://figshare.com/s/819dd80934cbc4f37564.

[7] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–12. https://doi.org/10.1109/ESEM.2019.8870181

[8] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories* (Shanghai, China) *(MSR '06)*. Association for Computing Machinery, New York, NY, USA, 137–143. https://doi.org/10.1145/1137983.1138016

[9] Tegawendé F. Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillère, Jacques Klein, and Yves Le Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 188–197. https://doi.org/10.1109/ISSRE.2013.6698918

[10] Fabio Calefato, Marco Aurélio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. 2022. Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub. *Empirical Softw. Engg.* 27, 3 (may 2022), 41 pages. https://doi.org/10.1007/s10664-021-10012-6

[11] Andrea Capiluppi, Klaas-Jan Stol, and Cornelia Boldyreff. 2011. Software Reuse in Open Source: A Case Study. *International Journal of Open Source Software and Processes* 3, 3 (2011), 10–35.

[12] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 186–196. https://doi.org/10.1145/3106237.3106246

[13] Jailton Coelho, Marco Tulio Valente, Luciana L. Silva, and André Hora. 2018. Why we engage in FLOSS: answers from core developers. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering* (Gothenburg, Sweden) *(CHASE '18)*. Association for Computing Machinery, New York, NY, USA, 114–121. https://doi.org/10.1145/3195836.3195848

[14] K. Crowston, Kangning Wei, Qing Li, and J. Howison. 2006. 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. 118a–118a. https://doi.org/10.1109/HICSS.2006.101

[15] Abdelkader Daghfous, Abroon Qazi, and M. Sajid Khan. 2021. Incorporating the risk of knowledge loss in supply chain risk management. *The International Journal of Logistics Management* (2021). https://api.semanticscholar.org/CorpusID:233712078

[16] Fabio Ferreira, Luciana Lourdes Silva, and Marco Tulio Valente. 2020. Turnover in open-source projects: The case of core developers. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*. 447–456.

[17] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. 2015. Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 829–841. https://doi.org/10.1145/2786805.2786870

[18] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. 2020. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *J. Syst. Softw.* 175 (2020), 110911. https://api.semanticscholar.org/CorpusID:222177507

[19] Philip Gray. 2022. To disengage or not to disengage: a look at contributor disengagement in open source software. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 328–330. https://doi.org/10.1145/3510454.3522685

[20] Giuseppe Iaffaldano, Igor Steinmacher, Fabio Calefato, Marco Gerosa, and Filippo Lanubile. 2019. Why do developers take breaks from contributing to OSS projects? a preliminary analysis. In *Proceedings of the 2nd International Workshop on Software Health* (Montreal, Quebec, Canada) *(SoHeal '19)*. IEEE Press, 9–16. https://dl.acm.org/doi/10.1109/SoHeal.2019.00009

[21] Glenn D Israel et al. 1992. Determining sample size. (1992).

[22] Ankur Joshi, Saket Kale, Satish Chandel, and D Kumar Pal. 2015. Likert scale: Explored and explained. *British journal of applied science & technology* 7, 4 (2015), 396–403.

[23] Rajdeep Kaur and Kuljit Chahal. 2022. Exploring factors affecting developer abandonment of open source software projects. *Journal of Software: Evolution and Process* 34 (06 2022). https://doi.org/10.1002/smr.2484

[24] Steve Klabnik and Carol Nichols. 2023. *The Rust programming language*. No Starch Press.

[25] Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart de Water. 2018. Studying pull request merges: a case study of shopify's active merchant. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice* (Gothenburg, Sweden) *(ICSE-SEIP '18)*. Association for Computing Machinery, New York, NY, USA, 124–133. https://doi.org/10.1145/3183519.3183542

[26] Erik Kouters, Bogdan Vasilescu, Mark G. J. van den Brand, and Alexander Serebrenik. 2012. Who's who in Gnome: Using LSA to merge software repository identities. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM) (ICSM '12)*. IEEE Computer Society, USA, 592–595. https://doi.org/10.1109/ICSM.2012.6405329

[27] Manny M Lehman. 1984. Program evolution. *Information Processing & Management* 20, 1-2 (1984), 19–36.

[28] Zhixing Li, Yue Yu, Tao Wang, Gang Yin, ShanShan Li, and Huaimin Wang. 2022. Are You Still Working on This? An Empirical Study on Pull Request Abandonment. *IEEE Transactions on Software Engineering* 48, 6 (2022), 2173–2188. https://doi.org/10.1109/TSE.2021.3053403

[29] Ju Long. 2006. Understanding the Role of Core Developers in Open Source Software Development. *Journal of Information, Information Technology, and Organizations (Years 1-3)* 1 (01 2006). https://doi.org/10.28945/148

[30] Jeffrey D. Long, Du Feng, and Norman Cliff. 2003. Ordinal Analysis of Behavioral Data. https://api.semanticscholar.org/CorpusID:123275712

[31] Courtney Miller, David Gray Widder, Christian Kästner, and Bogdan Vasilescu. 2019. Why Do People Give Up FLOSSing? A Study of Contributor Disengagement in Open Source. In *Open Source Systems*, Francis Bordeleau, Alberto Sillitti, Paulo Meirelles, and Valentina Lenarduzzi (Eds.). Springer International Publishing, Cham, 116–129.

[32] Felipe Curty do Rego Pinto and Leonardo Gresta Paulino Murta. 2023. On the assignment of commits to releases. *Empirical Softw. Engg.* 28, 2 (jan 2023), 35 pages. https://doi.org/10.1007/s10664-022-10263-x

[33] Xiangju Qin, Michael Salter-Townshend, and Padraig Cunningham. 2014. Exploring the Relationship between Membership Turnover and Productivity in Online Communities. *ArXiv* abs/1401.7890 (2014). https://api.semanticscholar.org/CorpusID:261908305

[34] Mehvish Rashid, Paul M. Clarke, and Rory V. O'Connor. 2017. Exploring Knowledge Loss in Open Source Software (OSS) Projects. In *Software Process Improvement and Capability Determination*, Antonia Mas, Antoni Mesquida, Rory V. O'Connor, Terry Rout, and Alec Dorling (Eds.). Springer International Publishing, Cham, 481–495.

[35] Mehvish Rashid, Paul M. Clarke, and Rory V. O'Connor. 2019. A systematic examination of knowledge loss in open source software projects. *Int. J. Inf. Manag.* 46, C (jun 2019), 104–123. https://doi.org/10.1016/j.ijinfomgt.2018.11.015

[36] Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, and Audris Mockus. 2016. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 1006–1016. https://doi.org/10.1145/2884781.2884851

[37] Martin P Robillard. 2021. Turnover-induced knowledge loss in practice. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1292–1302.

[38] Martin P. Robillard. 2021. Turnover-induced knowledge loss in practice. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1292–1302. https://doi.org/10.1145/3468264.3473923

[39] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Israel Herraiz. 2009. Evolution of the core team of developers in libre software projects. In *2009 6th IEEE International Working Conference on Mining Software Repositories*. 167–170. https://doi.org/10.1109/MSR.2009.5069497

[40] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices?. In *Annual Meeting of the Southern Association for Institutional Research*. 1–51.

[41] Bernard Rosner, Robert J Glynn, and Mei-Ling T Lee. 2006. The Wilcoxon signed rank test for paired comparisons of clustered data. *Biometrics* 62, 1 (2006), 185–192.

[42] Giuseppe Russo Latona, Christoph Gote, Christian Zingg, Giona Casiraghi, Luca Verginer, and Frank Schweitzer. 2024. Shock! Quantifying the Impact of Core Developers' Dropout on the Productivity of OSS Projects. In *Companion Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 706–709. https://doi.org/10.1145/3589335.3651559

[43] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. 2016. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. *Empirical Softw. Engg.* 21, 2 (apr 2016), 642–683. https://doi.org/10.1007/s10664-015-9406-4

[44] William Schueller, Johannes Wachs, Vito DP Servedio, Stefan Thurner, and Vittorio Loreto. 2022. Evolving collaboration, dependencies, and use in the rust open source software ecosystem. *Scientific Data* 9, 1 (2022), 703.
[45] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.
[46] R John Simes. 1986. An improved Bonferroni procedure for multiple tests of significance. *Biometrika* 73, 3 (1986), 751–754.
[47] Stack Overflow. 2023. 2023 Stack Overflow Developer Survey. https://survey.stackoverflow.co/2023/#technology-admired-and-desired.
[48] Synopsys. 2024. 2024 Open Source Security and Risk Analysis Report. https://www.synopsys.com/blogs/software-security/open-source-trends-ossra-report.html.
[49] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 2389–2401. https://doi.org/10.1145/3510003.3510205
[50] James Tizard, Peter Devine, Hechen Wang, and Kelly Blincoe. 2023. A Software Requirements Ecosystem: Linking Forum, Issue Tracker, and FAQs for Requirements Management. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2381–2393. https://doi.org/10.1109/TSE.2022.3219458
[51] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards Automating Code Review Activities. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 163–174. https://doi.org/10.1109/ICSE43902.2021.00027
[52] Ying Wang, Peng Sun, Lin Pei, Yue Yu, Chang Xu, Shing-Chi Cheung, Hai Yu, and Zhiliang Zhu. 2023. <sc>Plumber</sc>: Boosting the Propagation of Vulnerability Fixes in the <italic>npm</italic> Ecosystem. *IEEE Trans. Softw. Eng.* 49, 5 (may 2023), 3155–3181. https://doi.org/10.1109/TSE.2023.3243262
[53] Mairieli Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco A. Gerosa, and Igor Steinmacher. 2022. Bots for pull requests: the good, the bad, and the promising. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 274–286. https://doi.org/10.1145/3510003.3512765
[54] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 182 (nov 2018), 19 pages. https://doi.org/10.1145/3274451
[55] Zhou Yang, Chenyu Wang, Jieke Shi, Thong Hoang, Pavneet Kochhar, Qinghua Lu, Zhenchang Xing, and David Lo. 2023. What Do Users Ask in Open-Source AI Repositories? An Empirical Study of GitHub Issues. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. 79–91. https://doi.org/10.1109/MSR59073.2023.00024
[56] Yiqing Yu, Alexander Benlian, and Thomas Hess. 2012. An Empirical Study of Volunteer Members' Perceived Turnover in Open Source Software Projects. In *2012 45th Hawaii International Conference on System Sciences*. 3396–3405. https://doi.org/10.1109/HICSS.2012.97
[57] Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. 2024. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Transactions on Software Engineering* 50, 4 (2024), 816–835. https://doi.org/10.1109/TSE.2024.3364675
[58] Yuxia Zhang, Minghui Zhou, Audris Mockus, and Zhi Jin. 2019. Companies' participation in oss development–an empirical study of openstack. *IEEE Transactions on Software Engineering* 47, 10 (2019), 2242–2259.
[59] Jiaxin Zhu and Jun Wei. 2019. An Empirical Study of Multiple Names and Email Addresses in OSS Version Control Repositories. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 409–420. https://doi.org/10.1109/MSR.2019.00068

Received 2025-02-26; accepted 2025-04-01