

The dynamics of leadership and success in software development teams

Received: 6 May 2024

Accepted: 9 April 2025

Published online: 27 April 2025

Lorenzo Betti¹, Luca Gallo^{1,2}, Johannes Wachs^{3,4,5} & Federico Battiston¹✉

From science to industry, teamwork plays a crucial role in knowledge production and innovation. Most studies consider teams as static groups of individuals, thereby failing to capture how the micro-dynamics of collaborative processes and organizational changes determine team success. Here, we leverage fine-grained temporal data on software development teams from three software ecosystems – Rust, JavaScript, and Python – to gain insights into the dynamics of online collaborative projects. Our analysis reveals an uneven workload distribution in teams, with stronger heterogeneity correlated with higher success, and the early emergence of a lead developer carrying out the majority of work. Moreover, we find that a sizeable fraction of projects experience a change of lead developer, with such a transition being more likely in projects led by inexperienced users. Finally, we show that leadership change is associated with faster success growth. Our work contributes to a deeper understanding of the link between team evolution and success in collaborative processes.

The production of innovation and knowledge increasingly relies on collective efforts. For example, teamwork is crucial in scientific research, where teams have demonstrated their effectiveness in fostering groundbreaking discoveries^{1–3}, or in industry, where teamwork is essential for the rapid development of innovative solutions⁴. The effectiveness of teamwork depends on the continuous integration of specialized knowledge of individuals^{5,6}, and the ability to leverage constructive conflicts to generate novel insights^{7,8}. Extensive research has investigated characteristics of team members that are associated with performance⁹, from the effect of team size³, the interdependencies among team members¹⁰, and their diversity in terms of gender^{11–15}, expertise¹⁶, prior experiences^{2,17}, and ethnicity^{18,19}. In this way, the results of such collaborative efforts often go beyond the sum of individual contributions thanks to the emergence of synergies among team members^{20,21}, conditional on the team's organization.

In a team, not all members are equal²². Successful teamwork often relies on the management of tensions by leaders^{23,24}, who organize work

through the division of complex tasks into sub-tasks among different team members²⁵. Even in self-organizing teams, lacking a predefined hierarchical structure, specific team members can emerge as leading figures who carry out a sizeable fraction of the work and are eventually responsible for the project advancement^{26–28}. The organizational dynamic and evolving nature of teams impact team structure over time, as roles and responsibilities shift to adapt to new challenges^{29,30}. For instance, adjustments in the distribution of labor, the emergence and shift of leadership, and the turnover of team members all hinder coordination in teams, explaining why social and coordination skills are growing in value as teams get larger and teamwork more prevalent³¹.

Indeed, various conceptual frameworks have highlighted the need to explicitly consider team dynamics to properly understand how teams function and their success^{30,32,33}. Nevertheless, most of the insights we have consider teams as static entities, with little empirical research accounting for their temporal evolution. This is largely due to difficulties in accessing or collecting temporal data tracking and measuring team

¹Department of Network and Data Science, Central European University, Vienna, Austria. ²ANETI Lab, Corvinus Institute for Advanced Studies (CIAS), Corvinus University of Budapest, Budapest, Hungary. ³Institute for Data Analytics and Information Systems, Corvinus University of Budapest, Budapest, Hungary.

⁴Institute of Economics, HUN-REN Centre for Economic and Regional Studies, Budapest, Hungary. ⁵Complexity Science Hub, Vienna, Austria.

✉ e-mail: battistonf@ceu.edu

activities across time^{34,35}. In science, benefiting from the availability of large-scale curated publication data, a few studies have recently explored the concept of “persistent teams”³⁶ – researchers who consistently collaborate together over time – showing that “fresh” teams made of new collaborators produce more impactful research³⁷. However, publication data only records the outcome of teamwork, lacking information about the process of the collaborative effort, such as the specific contributions and activities of the authors of each paper. An exception is the analysis of email communications within research teams, which can provide insights into individual contributions and team dynamics³⁸.

In organization theory, team activity is typically tracked through multi-period observations, where team members are surveyed at various intervals. Yet this approach suffers from inherent limitations of low temporal resolution because it would necessitate frequently surveying team members. Such frequent surveys can disrupt teamwork and lead to survey fatigue, eventually compromising data quality and preventing the effective investigation of team dynamics at a microscopic scale³⁴. While fine-grained data about team processes can be collected through other methods like sensors that generate real-time data³⁹ or in controlled laboratory experiments⁴⁰, these approaches typically cover only a limited number of teams and are difficult to scale.

Open-source software development offers an ideal opportunity to investigate collaborative dynamics at a large scale and with a fine-grained resolution^{41–44}. In this context, software developers typically track changes to the software codebase using version control systems (e.g., Git) and store their software in online public repositories (e.g., GitHub). For instance, by analyzing the contribution history of a project via its commit log (a record of every edit made to code), it is possible to obtain information about who made which modification to the software and when. The analysis of this source of data has identified patterns where few developers often perform the majority of contributions in a project and most developers make few^{45–47}. This tendency towards centralization, widespread across collaborative software development, has been employed to develop heuristics to identify the core members of teams⁴⁸. Changes in the activity of core developers, as well as changes in its composition, can offer novel insights about how teams function and the relationship to project success⁴⁹. The dynamics of the main developers may vary significantly across projects. For instance, Linus Torvalds, who created the Linux kernel in 1991, still dominates the development effort of the project⁵⁰ and defines himself as a “benevolent dictator of Planet Linux”⁵¹. On the other hand, Pandas’ creator Wes McKinney stopped actively working on Pandas after five years⁵² and other developers have succeeded as the main developers of the project⁵³. While both Linux and the Pandas library are examples of successful projects, their contrasting leadership dynamics highlight a gap in our understanding of the causes and consequences of the emergence of key team members, their potential turnover, and the effect of such changes on project success.

To advance the understanding of team dynamics in open-source software projects, we analyze teams’ activity within three software ecosystems – Rust, JavaScript, and Python – along with their success metrics. First, we examine the distribution of work as reflected by the distribution of commits among team members, study the activity patterns of the lead developers – users responsible for the largest share of commits – and correlate it with project success. Then, we identify repositories where the lead developer changes during the project’s lifespan, identifying profound redistribution of workload and a potential reorganization of the team. Finally, we investigate the association between the change of lead developer and the success growth of the project after the transition by comparing those repositories to similar ones that did not change the lead developer. Our findings, consistent across the three software ecosystems, demonstrate the interplay of team dynamics and performance in open-source software projects, suggesting that changes in team organization have implications for the success of the project.

Results

We leverage three datasets tracking the activity of software developers working on GitHub repositories hosting packages from three popular programming languages: Rust, JavaScript, and Python. For Rust, we rely on a well-curated dataset covering over 40,000 developers working on more than 6000 repositories between 2014 and 2022, including their success time series^{54,55}. The JavaScript and Python datasets are about four and two times bigger, respectively, with activity from 2010 to 2018^{56,57}. We supplement these two datasets by collecting success time series data (see Supplementary Section S3 of the Supplementary Material).

In the main text, we present results focusing on the Rust ecosystem because it offers the greatest level of detail and coverage of the three systems. Indeed, as Rust was released in 2015, we have access to comprehensive temporal data on project development and GitHub-specific metrics such as stars. Moreover, the Rust package manager records download data since release. Javascript and Python, on the other hand, were created in the 1990s, well before the creation of GitHub, which limits our ability to collect comprehensive data on these languages. Similarly, data on downloads for Python and Javascript libraries are available only for limited time periods. Despite these limitations, we can still run our analyses in these much larger and longer-running systems and our substantive findings on team dynamics and project success are consistent across all three languages. We provide detailed information for JavaScript and Python in Supplementary Section S3 of the Supplementary Material.

Emergence of a lead developer

Software developers track their changes to the software codebase through commits, whose distribution across team members can measure their work contributions⁴⁷ and be informative about their roles in a project⁴⁸. After having verified that most of the commits refer to actual coding activities (see Supplementary Fig. S1)⁵⁸, we begin by analyzing the distribution of commits among team members to characterize the distribution of work in software development teams. We define a team as the set of developers who make at least one commit to a repository. By ranking the developers of a repository by their total number of commits, we measure the fraction of commits authored by developers as a function of their rank. Figure 1a shows that the most active developer (rank = 1) authors more than half of the total number of commits. In contrast, the second most active developer typically accounts for only around 10–20% of the total, while the rest is done by the other team members. This observation highlights the presence of a “lead developer” who carries the majority of the workload in a repository, alongside other developers contributing to a lesser extent. Those properties are consistent across teams of different sizes as shown in Fig. 1a (see Supplementary Fig. S2a for the distribution of repositories across different team sizes), through the lifetime of repositories (see Supplementary Fig. S6), and cannot be explained by random activity of developers (see Supplementary Fig. S4). This persistent nature of the workload distribution suggests a possible advantage from such a centralization.

One potential aspect where this advantage may manifest is projects’ success. To test for a relationship between heterogeneous workload distribution and success, we first quantify the heterogeneity of the workload distribution using the relative effective team size⁴⁷. This metric is defined as:

$$\text{rel. eff. team size} = 2^H / N$$

where H is the binary entropy of the distribution of commits among team members and N is the size of the team. This measure ranges from $1/N$ (i.e., one single developer makes all commits) to 1 (i.e., the workload is evenly distributed among all members). Consequently, a smaller relative effective team size indicates a more uneven

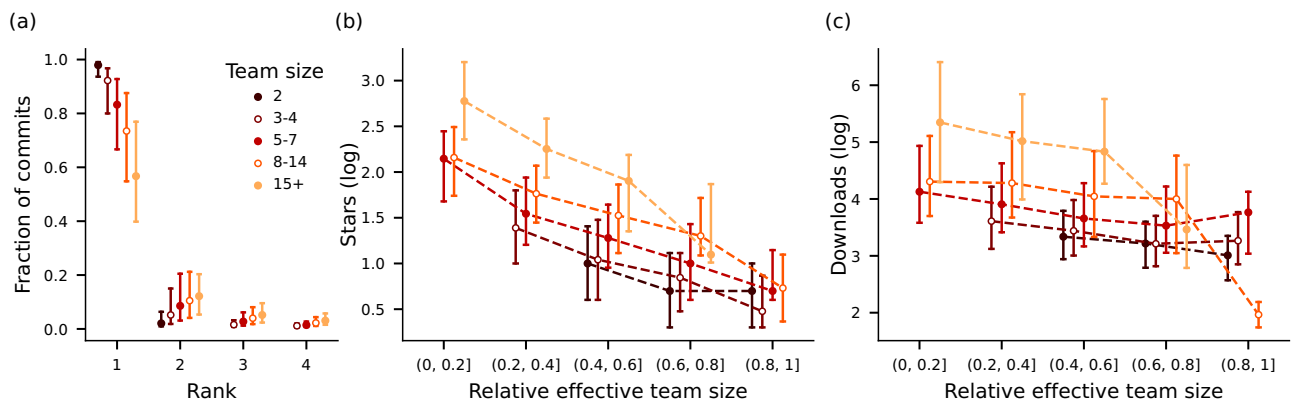


Fig. 1 | Workload distribution within teams and relationship with success.

a Median fraction of commits authored by the r -th most active developer of a repository stratified by team size. The most active developer makes more than half of the total number of commits while other developers contribute substantially less, regardless of the size of the team. Median number of stars **(b)** and downloads

(c) as a function of the relative effective team size stratified by team size. The more heterogeneous the workload distribution in the team, the higher the success. The Spearman's rank test returns $p < 0.001$ for all team sizes. The number of stars and downloads are incremented by one unit. Error bars range from the 25th to the 75th percentile of the distributions.

distribution of commits among team members. Then, we employ the number of stars and downloads as metrics of repositories' success. Such metrics track two different dimensions of success: stars can be considered as a proxy for repositories' popularity, similar to likes in social media⁵⁹, whereas the number of downloads reflects factors such as utility, necessity, and perceived quality. Our analysis reveals an inverse relationship between repositories' success and their relative effective team size, as displayed in Fig. 1b for the number of stars and Fig. 1c for the number of downloads (see Supplementary Fig. S2b for the distribution of the number of repositories for different relative effective team sizes). The relationship is supported by Spearman's correlation test, yielding a correlation coefficient of at least $\rho = -0.28$ for stars and $\rho = -0.11$ for downloads across all team sizes ($p < 0.001$; see Supplementary Table S1 for details). Note that we grouped repositories according to their team size since this is already strongly correlated with success (see Supplementary Fig. S5). To check if the age of repositories affects the results of the correlation, we repeated the analysis considering the relative effective team size and success at different moments of repositories' lifetime, finding consistent results (see Supplementary Fig. S6 and Supplementary Table S1).

Our observation aligns with prior research on software development teams⁴⁶, confirming that even in a relatively new programming language such as Rust, the most successful teams of software developers have an uneven workload distribution⁴⁷. Same findings hold for JavaScript and Python as shown in Supplementary Fig. S18. Furthermore, as already found in⁴⁵ where three large open-source projects were studied, our results show that the workload distribution becomes heterogeneous already within the first year of activity. Differently from previous studies, in the following we provide a characterization of the behaviors of the lead developers of each repository. Beyond static analyses, we focus in particular on team dynamics, revealing changes in workload distribution and how this impacts a project's success.

Characterization of lead developers' activity

Lead developers are not just prolific team members, but they fulfill specific management and coordination roles within their projects. The first aspect that underscores a distinguished role is whether they have direct write privileges, granting them the authority to integrate changes into the repository. We found that all identified lead developers do indeed have this right (see Supplementary Section S1.1 of the Supplementary Material). This privilege extends beyond integrating their own work, but it also confers the authority to review and decide whether others' contributions are worth being accepted. By searching for commits related to merging pull requests – the principal mechanism of

collaboration in GitHub – we revealed that most lead developers are heavily involved in such merging activities (83% of them; see Supplementary Section S1.2 of the Supplementary Material), suggesting that they review and decide which contributions to accept. Finally, coordinating team efforts is critical to managing a project effectively. We therefore assessed lead developers' roles in coordination by analyzing their centrality in the communication networks built from discussions under issues and pull requests, finding that they frequently occupy the central positions (in 82% of the cases they are among the top-three; see Supplementary Section S1.3 of the Supplementary Material). Taken together, these observations reveal that lead developers fulfill core leadership tasks, from integrating and reviewing external contributions to coordinating team efforts.

We now shift our focus to how the coding activities of leaders differ from those of other developers. We compare the distribution of their inter-commit time, defined as the time elapsed between two consecutive commits authored by the same user (regardless of the repository in which the commit is made). Figure 2a displays the distributions for lead and non-lead developers separately, showing distinct characteristics for those two sets of users. In particular, lead developers exhibit a higher frequency of commits. Indeed, their inter-commit time distribution is left-skewed compared to that of non-lead developers, displaying a prominent peak around 30 minutes (the peak in non-lead developers' inter-commit time ranges between one day and one week). This difference turns into lead developers being more likely to have longer streaks of consecutive commits, as depicted in the inset of Fig. 2a. The inset shows the probability distribution of making more than E consecutive commits whose inter-commit time is smaller than $\Delta t = 30$ min (the observation is robust across other values of Δt as can be seen in Supplementary Fig. S7). Lead developers are indeed three times more likely to make streaks of at least 10 commits ($P(\geq 10) = 0.0097$) compared to non-lead developers ($P(\geq 10) = 0.0030$).

Another aspect of the activity pattern of developers is the number of repositories they contribute to. We first show in Fig. 2b the distribution of the number of repositories in which lead and non-lead developers author at least one commit (see Supplementary Fig. S3 for the aggregate distribution). In addition to being the most active developers in their repositories, lead developers contribute to a median of 4 repositories (interquartile range IQR 2–8) while non-lead developers are active in only a median of 1 repository (IQR 1–2). This difference is deemed significant according to the Mann-Whitney U test ($U = 96197128$, $n_1 = 3627$ and $n_2 = 32998$, $p < 0.0001$). To gauge insights on the extent to which lead developers work on multiple projects simultaneously, we introduce the repository switch time of developers.

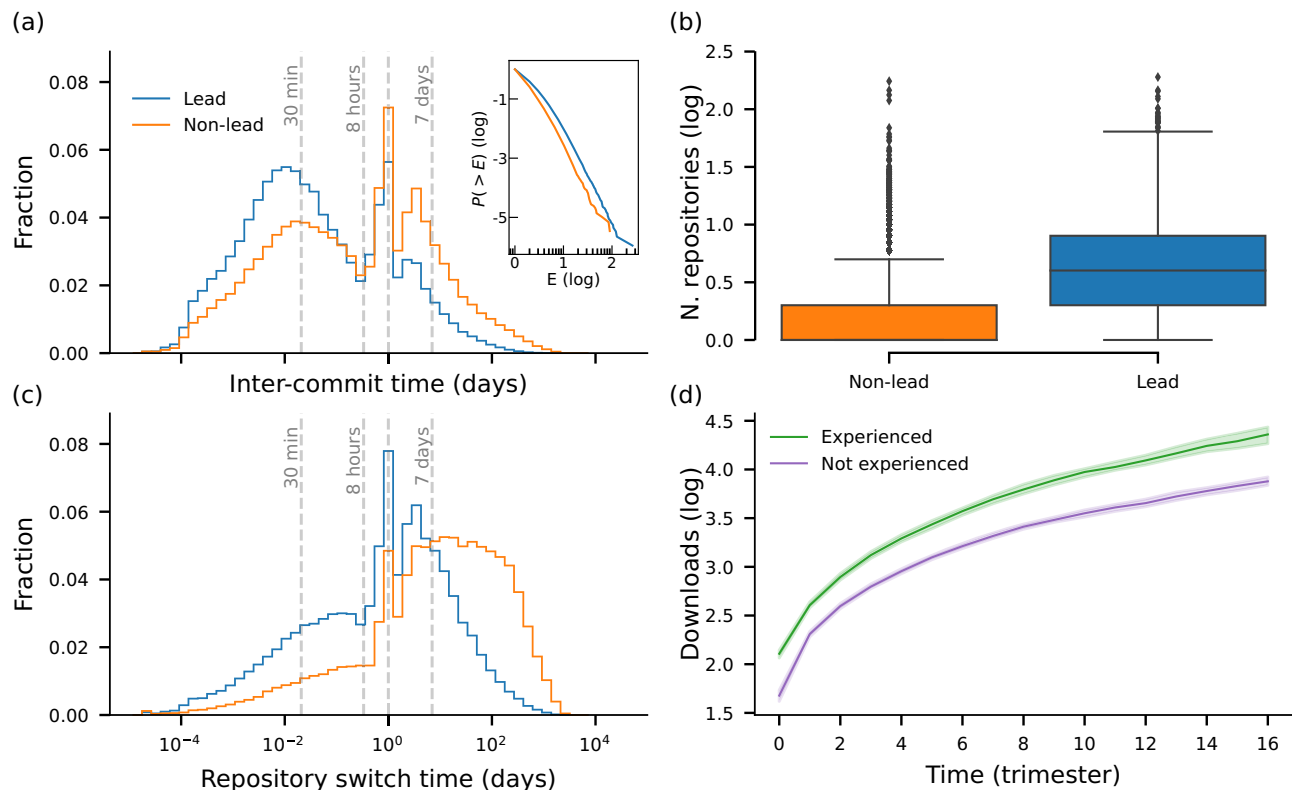


Fig. 2 | Characterization of lead developers' activity. **a** Distribution of the inter-commit times and cumulative distribution of the number of commits close in time (inset) for lead and non-lead developers. Lead developers exhibit higher frequency of commits and longer streaks of consecutive commits. **b** Distribution of the number of repositories in which lead and non-lead developers are active. Lead developers are involved in a larger number of repositories. Box plots indicate median (middle line), 25th, 75th percentile (box) and 5th and 95th percentile

(whiskers) as well as outliers (single points). **c** Distribution of the repository switch time of lead and non-lead developers. Lead developers tend to switch from one project to another on a daily to weekly basis. **d** Number of downloads across repositories' lifetime stratified by lead developers' experience (median and its 95% confidence interval). Repositories led by experienced developers are downloaded more compared to those led by inexperienced ones. Time is binned into trimesters.

This is defined as the time elapsed between a developer's first commit on a repository and their first commit on a different one, namely the time elapsed between a developer initiating work on one repository and transitioning to another. Figure 2c shows the distribution of repository switch time for lead and non-lead developers separately. The difference with the inter-commit time distribution in Fig. 2a suggests that the trains of consecutive commits shortly separated in time (i.e., inter-commit time of 30 min) are mostly done on the same repository, as lead developers switch from one repository to another on a daily to weekly basis. The rapid decay of the repository switch time after one week, in addition to the number of repositories in which lead developers are active, implies that they work concurrently on multiple projects and focus their efforts within the Rust ecosystem. Non-lead developers rather move between repositories over longer periods. This may be a consequence of non-lead developers being active in fewer projects and possibly being less involved in the Rust ecosystem.

Lead developers often contribute to multiple repositories, and their experience as core members in previous projects might prove beneficial for new endeavours. For this reason, we ask: Does an experienced lead developer impact the success of a repository? We define the lead developer of a repository i as experienced if they have been the lead developer of other repositories before starting to contribute to repository i . We identify 2744 (43%) repositories led by experienced lead developers and show in Fig. 2d the median number of downloads these repositories receive across their lifetime, compared to repositories whose lead developer never led a repository before. We observe that repositories led by developers with previous experience receive more downloads over their lifetime. Since

downloads reflect factors such as utility, necessity, and perceived quality, we speculate that developers who led repositories in the past may leverage their previous experience in Rust projects to develop software that is more useful and widely needed. By contrast, it is worth noticing that lead developers' previous experience does not affect the number of stars, indicating a lack of advantage in terms of popularity (see Supplementary Fig. S8). We verified that the observations are not influenced by the inclusion of repositories with varying ages, as this could introduce bias due to the advantage older repositories may have in accumulating stars and downloads (see Supplementary Fig. S8).

All these observations are mirrored in the other two datasets. Specifically, we show the distribution of inter-commit times in Supplementary Fig. S19a, e, the distribution of the number of repositories where developers are active in Supplementary Fig. S19b, f, the distribution of repository switch time in Supplementary Fig. S19c, g, and the success across time depending on the experience of the lead developer in Supplementary Fig. S19d, h.

Lead developers of repositories can change

Lead developers emerge early in a project and display distinct activity patterns. Does the same individual consistently maintain such a role, or does this figure change in time? To identify such changes, we first aggregate developer activities (i.e., cumulative number of commits) and the success of repositories (i.e., cumulative number of stars and downloads) by trimesters. Then, we consider the lead developer of a repository at trimester t as the one who has made the largest share of commits up to that time (we excluded 206 repositories from the subsequent analyses to ensure the inclusion of repositories displaying

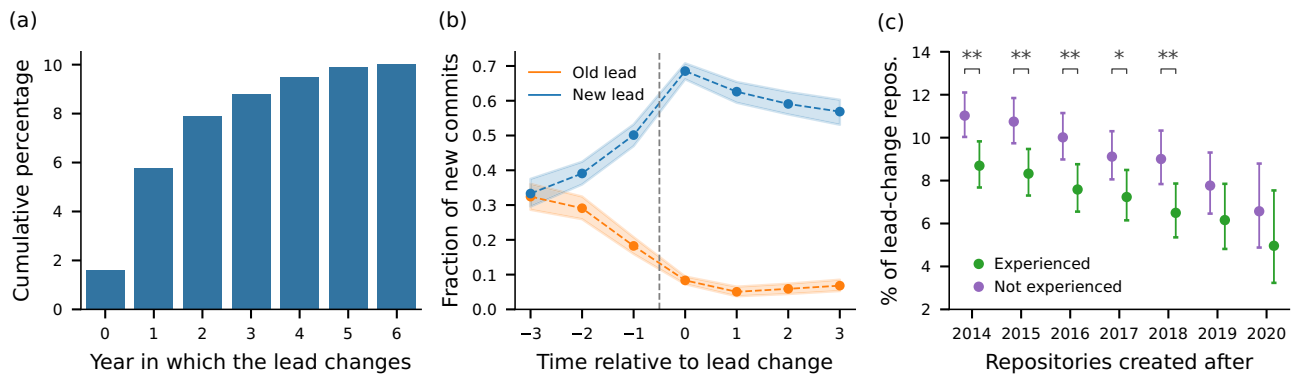


Fig. 3 | Lead developers can change across the lifetime of repositories.

a Cumulative percentage of repositories undergoing a lead developer change as a function of the number of years since their creation. Around 10% of repositories change their lead developer throughout their lifetime, with the majority occurring within the second and third year of activity. **b** Fraction of new commits authored by the old and new lead developer before and after the lead developer transition (mean and its 95% confidence interval). After the transition (vertical dashed line), contributions from the old lead developer diminish rapidly. **c** Percentage of lead-

change repositories stratified by the previous experience of the old lead developer. Each point refers to repositories created at a specific year or later. Repositories led by inexperienced lead developers exhibit a significantly higher likelihood to change their lead developer compared to those led by experienced ones, according to Fisher's exact test. Significance levels are denoted as follows: * for $p < 0.05$, ** for $p < 0.01$, and *** for $p < 0.001$. Error bars refer to 95% confidence intervals of the estimated percentages (Wilson score interval).

meaningful changes; see details in the Methods section). We find that 618 repositories (10%) undergo a change of lead developer, with the majority of such changes occurring between the second and third year of a repository's lifetime, as shown in Fig. 3a. Most repositories undergoing a change in lead developer experience only one transition (92%). Therefore, we focus our subsequent analyses on the first and second lead developers, hereafter referred to as the old and new lead developers, respectively.

To further investigate how the transition from the old to the new lead developer occurs, the first aspect we examine is the dynamics of the transition. We show in Fig. 3b the relative number of new commits authored by the old and new lead developer around the time when the transition occurs (i.e., the time when the lead developer changes), averaged across repositories (repositories with no activity in a trimester are omitted). The result indicates a rapid transition with a drop in the activity of the former lead developer. One year before the transition, old and new lead developers make a comparable amount of new contributions on average, i.e., 30% of the total number of new commits. However, while the activity of old lead developers sharply declines, dropping to levels below 10%, the relative contribution of new leads largely increases, ultimately making a heavy portion of new commits (around 65%) in the subsequent year after the change. In particular, in 297 repositories, corresponding to 48% of projects that undergo a change, the old lead developer ceases to contribute entirely through commits. This stop in commit activity does not necessarily indicate a complete departure from the project, as contributors can remain involved in other activities within projects. Indeed, we observe that in 48 of such repositories (16%), old lead developers remain active by engaging in other tasks such as addressing issues and reviewing pull requests. This transition of core team members to administrative roles has been previously observed in open-source software development as a response to an abrupt increase of external attention⁶⁰, and reflects in new lead developers moving to more central positions in the communication network of the project (see Supplementary Section S1.3 of the Supplementary Material).

Next, we ask if previous experience can explain changes to the role of the lead developer. To account for the temporal dimension, we consider the lead developer of a repository i as experienced if they have been the lead developer of other repositories before becoming the lead developer of repository i . We find that 11% of repositories initiated by inexperienced lead developers undergo a transition of lead developer (388 out of 3519), while this happens to 9% of those initiated by experienced lead developers (230 out of 2646). Although this

difference looks small, it corresponds to a 30% increase in the odds of changing the lead developer when the initial lead lacks previous experience and it is deemed significant according to Fisher's exact test (odds ratio at 1.30, p -value = 0.003). We checked the robustness of the result against the year in which repositories were initiated by conducting additional tests restricted to repositories initiated after specific years. The results, shown in Fig. 3c, confirm that the association remains significant except for the last two years, possibly because of the reduction of the sample size ($N = 2339$ for 2019 and $N = 1027$ for 2020). In short, the experience of the initial lead developer is associated with a lower likelihood of change.

Finally, we investigate if lead developers changes may be explained by the success of the project before the change. For instance, existing literature on startups suggests a U-shape relationship between founder departure and growth rate, indicating that founders of startups experiencing either slow or rapid growth are more likely to depart than those of startups with intermediate growth rates⁶¹. Interestingly, we find no evidence supporting an association between previous success and the likelihood of changing the lead developer in the future within the Rust ecosystem (see Supplementary Fig. S9 for details).

These results replicate almost entirely in the JavaScript and Python datasets. We provide the cumulative fraction of repositories undergoing a lead developer change in Supplementary Fig. S20a, d, the activity dynamics close to the transition point in Supplementary Fig. S20b, e, and the likelihood of lead developer change depending on the experience of the lead developer in Supplementary Fig. S20c, f. The main difference is the weaker yet significant association between lead developer change and experience in the JavaScript dataset (odds ratio at 1.11, $p = 0.02$).

Repositories that change the lead developer perform better after the change

With the previous analyses, we have shown that a sizeable fraction of repositories undergo a change of their lead developers, describing the dynamics and factors associated with this turnover. How does such a turnover relate to the repositories' future success?

To answer this question, we employ a matching approach to compare the success trend of those repositories (named "lead-change repositories") against the success of similar repositories whose lead developer did not change (named "lead-remain repositories"). Specifically, we design a stringed matching procedure to identify pairs of lead-change and lead-remain repositories that are similar in terms of

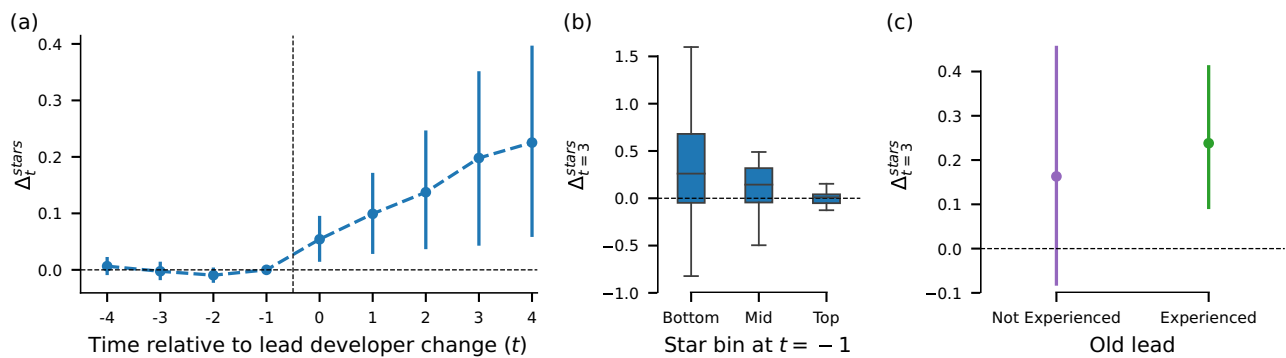


Fig. 4 | Lead developer changes are associated with faster success growth.

a Average effect of lead developer change Δ_t for stars. Repositories' success grows faster compared to similar repositories that did not undergo such a change. Success growth Δ_t for stars stratified by **b** the success before the change of lead developer and **c** the experience of the old lead developer. Worst-performing repositories exhibit a large positive effect following the change, whereas top-

performing ones are minimally affected. Repositories started by an experienced lead developer benefit more from the change than those initiated by an inexperienced one. Error bars in **(a)** and **(c)** correspond to 95% confidence intervals of the estimated quantities. In **(b)** box plots indicate median (middle line), 25th, 75th percentile (box) and 5th and 95th percentile (whiskers) as well as outliers (single points).

team composition and success prior to the change of the lead developer (see Methods section for details). To ensure a reasonable fit, we restrict the analysis to a subset of 151 lead-change repositories (24%) displaying meaningful activity and success one year before the change of lead developer (more than 50 commits, 10 stars, and 100 downloads). We then monitor the difference in the success growth at time t (Δ_t) associated to the lead developer change as

$$\Delta_t = Y_t - \widetilde{Y}_t$$

where $Y_t = S_t/S_{t_0}$ is the success of the lead-change repository (i.e., either number of stars or downloads) relative to the last pre-treatment period ($t_0 = -1$) and $\widetilde{Y}_t$ is that of the matched repository. The value of Δ_t should be close to zero for $t \leq t_0$, indicating that the lead-change repository and its match exhibit similar success trajectories before the change. After t_0 , Δ_t quantifies how the success growth varies in relation to the change of lead developer. We consider Δ_t for t ranging from $[-4, 4]$, namely, one year before and after the time in which the change occurs ($t = 0$).

We find that the change of lead developer is positively related to a stronger growth in success. Figure 4a shows the success difference Δ_t for stars averaged across the 135 repositories with a suitable match (89% of the lead-change repositories). Notably, the difference in success growth is already positive during the first trimester in which the new lead developer takes over ($\Delta_{t=0} = 0.05$ on average, 95% CI: 0.02–0.09) and keeps increasing during the whole year after the change (one year later, $\Delta_{t=4} = 0.23$ on average, 95% CI: 0.06–0.40). Same results hold for downloads (see Supplementary Fig. S13), thus suggesting that the change of lead developer is associated with faster success growth with respect to both popularity and perceived utility and quality of the software. The values of Δ_t before the change ($t < 0$) are close to zero, meaning that the success trends between the lead-change and their matched repositories are close on average before the transition. This underlines the good quality of the matching procedure. To check the robustness of the result, we also consider the median of the success difference Δ_t , showing that the observed positive association is not due to the skewness of the distribution of Δ_t (see Supplementary Fig. S13). Moreover, we verified that the observed association is not confounded by the heterogeneity in the skills and abilities of lead developers by performing a fixed effect regression, which yielded consistent results (see Supplementary Table S2). This finding is consistent for JavaScript and Python in both direction and magnitude, as shown in Supplementary Fig. S21a, d.

Finally, we investigate how success before the transition and lead developer experience affect success growth. As depicted in Fig. 4b, we

find that lowly successful repositories (i.e., those in the bottom 30% of the success distribution) have larger Δ_t in terms of number of stars when compared to average (i.e., those between the 50th and the 80th percentile) and top (i.e., top 10%) repositories (Kendall's $\tau_b = -0.18$, $p = 0.03$). This indicates that repositories gaining relatively more visibility after the change of lead developers are the least popular. We found no significant differences in the case of downloads (Kendall's $\tau_b = -0.01$, $p = 0.92$; see Supplementary Fig. S14). Regarding the previous experience, we observe in Fig. 4c that both repositories initiated by an experienced or not experienced lead developer have a significant growth in terms of stars (Not experienced: Wilcoxon's $T = 1682$, $n = 72$, $p = 0.02$, one sided; Experienced: Wilcoxon's $T = 1488$, $n = 63$, $p = 0.0004$ one sided) (see Supplementary Fig. S15 for a discussion of the skewness of Δ_t). Finally, in Supplementary Fig. S16, we consider the combined effect of both the old and the new lead developer's experience on success growth. We note that these observations are not in line with what found for JavaScript and Python. Specifically, we observe no difference in Δ_t for different levels of success before the transition (see Supplementary Fig. S21b, e) and mixed results for what concerns the differential effect of lead developers' experience on Δ_t (see Supplementary Fig. S21c, f). These discrepancies may be attributed to intrinsic differences of the software ecosystem under study.

Discussion

In this study, we used fine-grained data about software development teams from three popular programming languages to characterize the temporal dynamics of online collaborative projects. By tracking the activity of the most active contributor, we unveiled the emergence of a lead developer who exhibits a distinctive pattern of activity compared to non-lead developers, with lead developers emerging early in a project lifetime. Moreover, we show how an uneven distribution of workload among team members is positively associated with a repository's success. We identified a sizeable fraction of projects that undergo a change of their lead developer and revealed an association between such transitions and faster success growth.

Our analysis of open-source projects reports that most of the work is carried out by one or a few developers and that this correlates with higher success. This evidence is well-documented in the literature and spans across different programming languages^{45–47,62,63}. One possible explanation for such a correlation is the increase in efficiency due to the concentration of workload among few developers, which is likely to reduce the cost of coordination^{47,64}. At the same time, this distribution of work may result in a concentration of knowledge about the functioning of the software around a few developers, thus posing the project at risk should the main developers leave. Drawing on a well-

known concept in the software development literature, projects have a small “truck factor”, meaning that the number of key developers who would need to be incapacitated, i.e., hit by a truck, to prevent further development of the project is small^{65–69}. The tension between efficiency and project sustainability highlights a “high risk, high reward” strategy in open-source software development.

Moving beyond the open-source software development literature, whether a hierarchical or flatter organization fosters better team outcomes is a long-standing debate, summarized by the tension between the functionalist and the conflict perspectives. The former posits that team outcomes benefit from a hierarchical structure because it facilitates coordination, whereas the latter predicts a negative effect because of tensions arising from non-aligned goals across the levels of the hierarchy⁷⁰. Many empirical and theoretical efforts further demonstrate that such a relationship may depend on task type, task complexity, and additional mediators^{70–75}. These cross-cutting observations underscore the lack of an overarching understanding of this relationship and motivate further comparative studies to examine the interplay of domain specifics, team dynamics, and task type on team outcomes. Our findings add to this wider conversation by focusing on open-source development, where uneven team structures appear widespread and are linked to higher success. Notably, recent analyses of top-starred GitHub repositories indicate that highly successful projects often exhibit pronounced hierarchical collaboration structures, suggesting that such patterns of division of labor can indeed thrive in large, complex projects⁷⁶. Such analyses focus only on a few, large projects, and are difficult to scale⁷⁷. While our broader, commit-level approach enables an extensive coverage of projects, we acknowledge that it does not allow to detect these nuanced hierarchies or reveal the specialized or modular collaboration patterns that may emerge in the largest teams.

More broadly, our study contributes to the ongoing discourse within the field of team science, particularly by addressing the unresolved question of how changes in team composition affect teamwork³⁴. Although the literature presents mixed findings, where team changes can be either beneficial^{78,79} or detrimental^{80,81}, there is consensus that changes involving core team members deeply affect team functionality²². Our results are not only in line with these general predictions, but lean towards a positive effect of such changes on team outcomes. In addition, the stronger effect observed for experienced lead developers of Rust projects leaves room to speculate about the importance of experienced developers in setting the stage for successful projects regardless of potential changes in such a fundamental role. This observation raises further questions about the characteristics of the old and new lead developers that may influence such a relationship. Drawing on previous studies, the change of a team member is more beneficial the higher the relative skills of the team members involved in the turnover⁸². Further research is needed to identify factors that make the turnover more beneficial for team outcomes. Additionally, our study mainly focus on the figure of the lead developer but studying the turnover and eventual shifts in workload distribution among the other team members remains an interesting direction for future research.

While our findings relate the change of the lead developer with a significantly higher success growth, the mechanism behind this association remains unclear. Indeed, this relationship highlights a deep connection between the dynamics of teams and their performance, which leaves room for speculation on whether success drives leadership change or vice versa. For instance, as a result of a rapid increase of attention, projects can attract new contributors, and core developers can shift to organizational roles⁶⁰. This suggests that an increase in success brings more developers to join the team, thus increasing the probability of changing the lead developer. Since we consider three popular programming languages, a similar mechanism may be in place, which likely increases the number of new contributors to

projects and may require teams to adapt and undergo structural changes⁸³. Conversely, leadership turnover can unleash creative forces previously bound up in an organization or team, which drive the increase in the project’s success⁸⁴. At the same time, in our context we saw that new leaders tend to ramp up their activity in a project before taking over, suggesting that there may be value in a balance of previous collaborative ties and new connections in an evolving team²⁴. In short, further research is needed to shed light on the causality between lead developer change and success.

Even though we analyzed three programming languages, we chose to keep their analyses separate and focus on Rust in the main text. Besides the superior quality of the Rust dataset compared to the JavaScript and Python ones, we believe that our analyses benefit from considering the whole development history of one single programming language at a time, providing a controlled environment to characterize team dynamics in open-source software development. Indeed, considering multiple programming languages together may affect the results because coding practices differ depending on the programming language. For instance, it has been shown that different programming languages have varying levels of productivity and require unequal effort to write the same code^{85,86}. In addition, we cannot have access to complete success time series for relatively old programming languages such as JavaScript and Python. Indeed, the success metrics we considered (i.e., GitHub’s stars and downloads) are platform-specific and may not be available for programming languages that are older than software-development platforms, as we may not have data on their repositories’ entire development history. Keeping these challenges and limitations in mind, we have been able to compare such ecosystems obtaining similar results (see Supplementary Section S3 of the Supplementary Material). The generalizability of our findings to three among the largest and most used programming languages indicates that our results may apply to OSS development in general.

Beyond open-source software development, our work provides a fresh perspective on team evolution, leadership dynamics and their relationship to project success, contributing to a deeper understanding of the successful dynamics of collaborative processes.

Methods

In this section, we focus again on the Rust dataset for the sake of consistency with the main text. All the methods were applied consistently to the JavaScript and Python datasets. The major differences regard the construction of the JavaScript and Python datasets, which are detailed in Supplementary Section 3.1 of the Supplementary Material.

Data and selection of repositories

We used data sourced from ref. 54, consisting of a curated dataset containing the activities of developers across 39,671 repositories hosting Rust packages on various online platforms (e.g., GitHub, GitLab). Developers’ activities are tracked through commits, providing a comprehensive record of changes to the project’s codebase, in addition to other activities pertaining more to project management tasks (e.g., pull requests, Q&As). Developers’ user names are disambiguated, and flags identifying bot accounts are provided. The dataset includes platform-specific features that can be used as a proxy for repositories’ success over time. We chose to use the number of stars, which can be considered the most reliable measure of popularity⁵⁹, and number of downloads, which reflects factors such as utility, necessity, and perceived quality. Stars and downloads provide a multifaceted perspective on the success of repositories. Since stars are only available for repositories stored on GitHub, we discarded those hosted in other platforms (6% of the projects in the datasets).

Inspired by⁷⁷, we filtered repositories to ensure the inclusion of repositories suitable to study collaborative software development.

After discarding the activity of bots, we selected the repositories satisfying the following conditions: (1) first commit with no deletions, (2) total number of lines of code positive across the whole lifetime, (3) first commit in 2014 or later, (4) at least 100 lines of code in total, (5) lifetime of at least one year, (6) at least one commit per month on average, (7) at least one package associated to the repository (since a repository can host more than one package⁵⁴). Conditions (1) and (2) make us more confident that we study repositories for which their whole history is tracked. Indeed, no repository can be initiated by deleting any line, nor can it have a negative number of lines at any point in its lifetime. Conditions (4–7) select repositories hosting software developed over time and likely discard very small projects. After discarding repositories developed by one single developer, and repositories displaying activity on less than four trimesters, we ended up with a total of 6165 repositories.

Detecting lead developer changes

The lead developer of a repository is defined as the developer with the highest number of commits. However, different developers may be leading the repository at different points in time. To identify changes in lead developers over time, we counted the cumulative number of commits authored by team members at each time period (i.e., trimesters). Then, we defined the lead developer of the repository at time t as the team member that made the largest share of commits up to that time. To avoid the inclusion of spurious changes, such as those in the initial phases of the project when the activity is relatively low, we excluded the repositories that have changed the lead developer within the first three trimesters of activity. Additionally, cases where a lead developer change is followed by the former lead developer re-assuming their role were excluded as well (total repositories discarded $N_{\text{spurious}} = 206$). This criterion ensures that our analyses rely on repositories in which a meaningful change happened, with the initial lead developer maintaining their position for a significant duration before being succeeded by a new lead developer.

We describe in Supplementary Section S2 of the Supplementary Material a more restrictive definition of lead developer that introduces a statistical test to determine whether the most active developer has a commit count that is significantly larger than the one of the second lead developers. Our findings result robust to this alternative definition.

Matching procedure

To investigate the performance of repositories after the change of their lead developer ("lead-change repositories"), we compared their success trajectory with that of similar ones whose lead developer did not change ("lead-remain repositories"). Specifically, we implemented a matching procedure that, for each lead-change repository, identifies a set of lead-remain repositories that are similar to the lead-change repository in terms of temporal patterns of activity, team composition and success before the change of lead developer happened. Then, we selected the matched repository among those candidates as the one with the most similar success trajectory during the year preceding the change of lead developer.

The details of the matching are as follows. Initially, for each lead-change repository, we identified lead-remain repositories whose first commit date differs by at most six months from that of the lead-change repository. In addition, we required the lead-remain repositories to have lifetime as long as the age at which the lead-change repository changed the lead developer. This ensures that the lead-change repository and its candidates developed within a comparable timeframe. Such a requirement contributes to controlling for the average status of Rust's ecosystem, thus avoiding potential biases due to the rapid growth of the programming language. In this setting, $t = 0$ designates the time in which the new lead developer takes over. Then, we refined our set of candidates to select repositories that closely matched the lead-change repository in terms of team composition and prior

success for $t < 0$. The selected set of candidates met the following criteria: (1) similar team size at $t = -1$ (see Supplementary Fig. S12), (2) absolute difference in relative effective team size, averaged across $t \in [-4, -1]$, smaller than 0.20, and (3) absolute relative difference of (log) number of stars and downloads at $t = -1$ smaller than 0.50.

As the last step, we selected the candidate that most closely matched in terms of success growth before the change. To do that, we first defined the success growth as $Y_t = S_t/S_{t_0}$, where $t_0 = -1$ refers to the last time period before the change of the lead developer and S_t is either the number of stars or downloads at time t . In other words, we considered the success growth relative to the success of the repository at the last trimester before the lead developer change. We then chose the matched repository as the one that exhibited the smallest maximum relative difference in success growth during the period $t \in [-4, -1]$, considering both stars and downloads. If the maximum difference is larger than 0.50, we discarded the lead-change repository due to the lack of a sufficiently similar repository among the lead-remain ones. Our results are robust against the choice of those thresholds.

Data availability

All data associated with this manuscript are available on Zenodo: <https://zenodo.org/records/13837606>.

Code availability

All scripts associated with this manuscript are available on Zenodo: <https://zenodo.org/records/13837606>.

References

1. Wuchty, S., Jones, B. F. & Uzzi, B. The increasing dominance of teams in production of knowledge. *Science* **316**, 1036–1039 (2007).
2. Guimerà, R., Uzzi, B., Spiro, J. & Amaral, L. A. N. Team assembly mechanisms determine collaboration network structure and team performance. *Science* **308**, 697–702 (2005).
3. Wu, L., Wang, D. & Evans, J. A. Large teams develop and small teams disrupt science and technology. *Nature* **566**, 378–382 (2019).
4. Liu, M. -L., Liu, N. -T., Ding, C. G. & Lin, C. -P. Exploring team performance in high-tech industries: Future trends of building up teamwork. *Technol. Forecast. Soc. Change* **91**, 295–310 (2015).
5. Ben-Menahem, S. M., von Krogh, G., Erden, Z. & Schneider, A. Coordinating knowledge creation in multidisciplinary teams: Evidence from early-stage drug discovery. *Acad. Manag. J.* **59**, 1308–1338 (2016).
6. Grand, J. A., Braun, M. T., Kuljanin, G., Kozlowski, S. W. J. & Chao, G. T. The dynamics of team cognition: A process-oriented theory of knowledge emergence in teams. *J. Appl Psychol.* **101**, 1353–1385 (2016).
7. Xie, X. -Y., Wang, W. -L. & Luan, K. It is not what we have, but how we use it: Reexploring the relationship between task conflict and team innovation from the resource-based view. *Group Process. Inter-group Relat.* **17**, 240–251 (2014).
8. Thomas A. O'Neill, N. J. A. & Hastings, S. E. Examining the -pros- and -cons- of team conflict: A team-level meta-analysis of task, relationship, and process conflict. *Hum. Perform.* **26**, 236–260 (2013).
9. Mariani, M. S. et al. Collective dynamics behind success. *Nat. Commun.* **15**, 10701 (2024).
10. Baumann, F., Czaplicka, A. & Rahwan, I. Network structure shapes the impact of diversity in collective learning. *Sci. Rep.* **14**, 2491 (2024).
11. Yang, Y., Tian, T. Y., Woodruff, T. K., Jones, B. F. & Uzzi, B. Gender-diverse teams produce more novel and higher-impact scientific ideas. *Proc. Natl Acad. Sci.* **119**, e2200841119 (2022).
12. Bear, J. B. & Woolley, A. W. The role of gender in team collaboration and performance. *Interdiscip. Sci. Rev.* **36**, 146–153 (2011).
13. Wallrich, L. et al. The relationship between team diversity and team performance: reconciling promise and reality through a

- comprehensive meta-analysis registered report. *J. Bus. Psychol.* **39**, 1303–1354 (2024).
14. Zeng, X. H. T. et al. Differences in collaboration patterns across discipline, career stage, and gender. *PLOS Biol.* **14**, 1–19 (2016).
 15. Duch, J. et al. The possible role of resource requirements and academic career-choice risk on gender differences in publication rate and impact. *PLOS ONE* **7**, 1–11 (2012).
 16. de Vaan, M., Vedres, B. & Stark, D. Game changer: The topology of creativity. *Am. J. Sociol.* **120**, 1144–1194 (2015).
 17. Pobiedina, N., Neidhardt, J., Calatrava Moreno, M.D.C. and Werthner, H. Ranking factors of team success. In: *Proceedings of the 22nd International Conference on World Wide Web, WWW'13 Companion* (Association for Computing Machinery, 2013) p. 1185–1194.
 18. Freeman, R. B. & Huang, W. Collaborating with people like me: Ethnic coauthorship within the united states. *J. Labor Econ.* **33**, S289–S318 (2015).
 19. Hoogendoorn, S. & van Praag, M. Ethnic diversity and team performance: a field experiment. Discussion Paper 12-068/3 (Tinbergen Institute, 2012).
 20. Woolley, AnitaWilliams, Chabris, C. F., Pentland, A., Hashmi, N. & Malone, T. W. Evidence for a collective intelligence factor in the performance of human groups. *Science* **330**, 686–688 (2010).
 21. Ungar, L. et al. The good judgment project: A large scale test of different methods of combining expert predictions. In: *AAAI Fall Symposium: Machine Aggregation of Human Judgment* (AAAI, 2012).
 22. Arrow, H. & McGrath, J. E. Membership matters: How member change and continuity affect small group structure, process, and performance. *Small Group Res.* **24**, 334–361 (1993).
 23. Saeed, T., Almas, S., Anis-ul Haq, M. & Niazi, G. S. K. Leadership styles: relationship with conflict management styles. *Int. J. Confl. Manag.* **25**, 214–225 (2014).
 24. Vedres, B. & Cserpes, T. Open ties and deep success. In *Academy of Management Proceedings*, vol. 2020 (Academy of Management Briarcliff Manor, 10510, 2020) p. 18879.
 25. Zaccaro, S. J., Rittman, A. L. & Marks, M. A. Team leadership. *Leadersh. Q.* **12**, 451–483 (2001).
 26. O'Mahony, S. & Ferraro, F. The emergence of governance in an open source community. *Acad. Manag. J.* **50**, 1079–1106 (2007).
 27. Thapa, R. & Vidolov, S. Evaluating distributed leadership in open source software communities, ECIS 2020 Research Papers https://aisel.aisnet.org/ecis2020_rp/196 (2020).
 28. Taggar, S., Hackew, R. & Saha, S. Leadership emergence in autonomous work teams: Antecedents and outcomes. *Pers. Psychol.* **52**, 899–926 (1999).
 29. Simon, H. A. *Administrative behavior* (Simon and Schuster, 2013).
 30. Arrow, H., McGrath, J.E. & Berdahl, J.L. *Small Groups as Complex Systems: Formation, Coordination, Development, and Adaptation* (Sage Publications, 2000).
 31. Deming, D. J. The growing importance of social skills in the labor market. *Q. J. Econ.* **132**, 1593–1640 (2017).
 32. Delice, F., Rousseau, M. & Feitosa, J. Advancing teams research: What, when, and how to measure team dynamics over time. *Front. Psychol.* **10**, 1324 (2019).
 33. Gorman, J. C., Dunbar, T. A., Grimm, D. & Gipson, C. L. Understanding and modeling teams as dynamical systems. *Front. Psychol.* **8**, 1053 (2017).
 34. Humphrey, S. E. & Aime, F. Team microdynamics: Toward an organizing approach to teamwork. *Acad. Manag. Ann.* **8**, 443–503 (2014).
 35. Santolini, M. et al. iGEM: a model system for team science and innovation. Preprint at <https://arxiv.org/abs/2310.19858> (2023).
 36. Chowdhary, S., Gallo, L., Musciotto, F. & Battiston, F. Team careers in science: formation, composition and success of persistent collaborations. <http://arxiv.org/abs/2407.09326> [physics.soc-ph] (2024).
 37. Zeng, A., Fan, Y., Di, Z., Wang, Y. & Havlin, S. Fresh teams are associated with original and multidisciplinary research. *Nat. Hum. Behav.* **5**, 1314–1322 (2021).
 38. Duch, J., Waitzman, J. S. & Amaral, LuísA. Nunes Quantifying the performance of individual players in a team activity. *PLOS ONE* **5**, 1–7 (2010).
 39. Kolbe, M. & Boos, M. Laborious but elaborate: The benefits of really studying team dynamics. *Front. Psychol.* **10**, 1478 (2019).
 40. O. Szabo, R., Chowdhary, S., Deritei, D. & Battiston, F. The anatomy of social dynamics in escape rooms. *Sci. Rep.* **12**, 10498 (2022).
 41. Scholtes, I., Mavrodiev, P. & Schweitzer, F. From aristotle to ringelmann: a large-scale analysis of team productivity and coordination in open source software projects. *Empir. Softw. Eng.* **21**, 642–683 (2016).
 42. Gote, C., Scholtes, I. & Schweitzer, F. Analysing time-stamped co-editing networks in software development teams using git2net. *Empir. Softw. Eng.* **26**, 75 (2021).
 43. Sornette, D., Maillart, T. & Ghezzi, G. How much is the whole really more than the sum of its parts? $1 \boxplus 1 = 2.5$: Superlinear productivity in collective group actions. *PLOS ONE* **9**, e103023 (2014).
 44. Zöller, N., Morgan, J. H. & Schröder, T. A topology of groups: What github can tell us about online collaboration. *Technol. Forecast. Soc. Change* **161**, 120291 (2020).
 45. Goeminne, M. & Mens, T. Evidence for the pareto principle in open source software activity. In: *Proceedings of the 1st International workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability* (Citeseer, 2011) pp. 74–82.
 46. Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E. & Ubayashi, N. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In: *Proceedings of the 14th International Workshop on Principles of Software Evolution, IWPSE 2015* (Association for Computing Machinery, New York, NY, USA, 2015) p. 46–55.
 47. Klug, M. & Bagrow, J. P. Understanding the group dynamics and success of teams. *R. Soc. Open Sci.* **3**, 160007 (2016).
 48. Bock, T., Alznauer, N., Joblin, M. & Apel, S. Automatic core-developer identification on github: A validation study. *ACM Trans. Softw. Eng. Methodol.* **32**, 1–29 (2023).
 49. Zanetti, M. S., Scholtes, I., Tessone, C. J. & Schweitzer, F. The rise and fall of a central contributor: Dynamics of social organization and performance in the gentoo community. In *2013 6th international workshop on cooperative and human aspects of software engineering (chase)* (IEEE, 2013) pp. 49–56.
 50. Linuk kernel commit distribution on github, <https://web.archive.org/web/20240324122230/https://github.com/torvalds/linux/graphs/contributors> (2024).
 51. Rivlin, G. Leader of the free world, <https://www.wired.com/2003/11/linus/> (2003).
 52. Turck, M. In conversation with wes mckinney, ceo, ura computing, <https://mattturck.com/mckinney/> (2021).
 53. Pandas commit distribution on github, <https://web.archive.org/web/20240417080730/https://github.com/pandas-dev/pandas/graphs/contributors> (Accessed on 2024).
 54. Schueller, W., Wachs, J., Servedio, VitoD. P., Thurner, S. & Loreto, V. Evolving collaboration, dependencies, and use in the rust open source software ecosystem. *Sci. Data* **9**, 703 (2022).
 55. Schueller, W. & Wachs, J. Modeling interconnected social and technical risks in open source software ecosystems. *Collective Intell.* **3**, 26339137241231912 (2024).
 56. Valiev, M., Vasilescu, B. & Herbsleb, J. Ecosystem-level determinants of sustained activity in open-source projects: a case study of the pypi ecosystem. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and*

- Symposium on the Foundations of Software Engineering, ESEC/FSE 2018 (Association for Computing Machinery, 2018) p. 644–655.
57. Valiev, M., Vasilescu, B. & Herbsleb, J. Ecosystem-Level Factors Affecting the Survival of Open-Source Projects: A Case Study of the PyPI Ecosystem - the dataset, <https://doi.org/10.5281/zenodo.1297925> (2018).
 58. Hattori, L.P. & Lanza, M. On the nature of commits. In: *23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops* pp. 63–71 (IEEE, 2008).
 59. Borges, H. & Tulio Valente, M. What's in a github star? understanding repository starring practices in a social coding platform. *J. Syst. Softw.* **146**, 112–129 (2018).
 60. Maldeniya, D., Budak, C., Robert Jr, L. P. & Romero, D. M. Herding a deluge of good samaritans: How github projects respond to increased attention. In: *Proceedings of The Web Conference 2020* (Association for Computing Machinery, 2020) p. 2055–2065.
 61. Boeker, W. & Karichalil, R. Entrepreneurial transitions: Factors influencing founder departure. *Acad. Manag. J.* **45**, 818–826 (2002).
 62. Mockus, A., Fielding, R. T. & Herbsleb, J. D. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.* **11**, 309–346 (2002).
 63. Chetkowski, T., Gloor, P. & Jemielniak, D. Inequalities in open source software development: Analysis of contributor's commits in apache software foundation projects. *PLOS ONE* **11**, 1–19 (2016).
 64. Ye, Y., Nakakoji, K. & Yamamoto, Y. Reducing the cost of communication and coordination in distributed software development. In: *Software Engineering Approaches for Offshore and Outsourced Development*, edited by (eds Meyer, B. & Joseph, M.) (Springer, 2007) pp. 152–169.
 65. Avelino, G., Passos, L., Hora, A. & Valente, M. A novel approach for estimating truck factors. In: *IEEE 24th International Conference on Program Comprehension (ICPC)* (IEEE Computer Society, 2016) pp. 1–10.
 66. Pfeiffer, R.-H. Identifying critical projects via pagerank and truck factor. In: *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)* pp. 41–45 (IEEE, 2021).
 67. Williams, L. and Kessler, R. R. *Pair programming illuminated* (Addison-Wesley Professional, 2003).
 68. Bowler, M. Truck Factor, <https://web.archive.org/web/20210429072452/http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/> (2005).
 69. Ferreira, M., Mombach, T., Valente, M. T. & Ferreira, K. Algorithms for estimating truck factors: a comparative study. *Softw. Qual. J.* **27**, 1583–1617 (2019).
 70. Greer, L. L., de Jong, B. A., Schouten, M. E. & Dannals, J. E. Why and when hierarchy impacts team effectiveness: A meta-analytic integration. *J. Appl. Psychol.* **103**, 591–613 (2018).
 71. Brackbill, D. & Centola, D. Impact of network structure on collective learning: An experimental study in a data science competition. *PLOS ONE* **15**, 1–13 (2020).
 72. Almaatouq, A., Alsobay, M., Yin, M. & Watts, D. J. Task complexity moderates group synergy. *Proc. Natl Acad. Sci.* **118**, e2101062118 (2021).
 73. Lazer, D. & Friedman, A. The network structure of exploration and exploitation. *Adm. Sci. Q.* **52**, 667–694 (2007).
 74. Barkoczi, D. & Galesic, M. Social learning strategies modify the effect of network structure on group performance. *Nat. Commun.* **7**, 13109 (2016).
 75. Centola, D. The network science of collective intelligence. *Trends Cogn. Sci.* **26**, 923–941 (2022).
 76. Palazzi, M. J., Cabot, J., Cánovas Izquierdo, J. L., Solé-Ribalta, A. & Borge-Holthoefer, J. Online division of labour: emergent structures in open source software. *Sci. Rep.* **9**, 13890 (2019).
 77. Gote, C., Mavrodiev, P., Schweitzer, F. & Scholtes, I. Big data = big insights? operationalising brooks' law in a massive github data set. In: *Proceedings of the 44th International Conference on Software Engineering, ICSE '22* (Association for Computing Machinery, 2022) p. 262–273.
 78. Gorman, J. C., Amazeen, P. G. & Cooke, N. J. Team coordination dynamics. *Nonlinear Dyn. Psychol. Life Sci.* **14**, 265 (2010).
 79. Choi, H. -S. & Thompson, L. Old wine in a new bottle: Impact of membership change on group creativity. *Organ. Behav. Hum. Decis. Process.* **98**, 121–132 (2005).
 80. Pedro J. Ramos-Villagrasa, José Navarro & García-Izquierdo, A. L. Chaotic dynamics and team effectiveness: Evidence from professional basketball. *Eur. J. Work Organ. Psychol.* **21**, 778–802 (2012).
 81. Lewis, K., Belliveau, M., Herndon, B. & Keller, J. Group cognition, membership change, and performance: Investigating the benefits and detriments of collective knowledge. *Organ. Behav. Hum. Decis. Process.* **103**, 159–178 (2007).
 82. Summers, J. K., Humphrey, S. E. & Ferris, G. R. Team member change, flux in coordination, and performance: Effects of strategic core roles, information transfer, and cognitive ability. *Acad. Manag. J.* **55**, 314–338 (2012).
 83. Galesic, M. et al. Beyond collective intelligence: Collective adaptation. *J. R. Soc. Interface* **20**, 20220736 (2023).
 84. Tzabbar, D. & Kehoe, R. R. Can opportunity emerge from disarray? an examination of exploration and exploitation following star scientist turnover. *J. Manag.* **40**, 449–482 (2014).
 85. Lavazza, L., Morasca, S. & Tosi, D. An empirical study on the effect of programming languages on productivity. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16* (Association for Computing Machinery, 2016) p. 1434–1439.
 86. Prechelt, L. An empirical comparison of seven programming languages. *Computer* **33**, 23–29 (2000).

Acknowledgements

L.G. and F.B. acknowledge support from the Air Force Office of Scientific Research under award number FA8655-22-1-7025. J.W. acknowledges funding from the Hungarian National Scientific Fund (OTKA FK 145960) and from the European Union under Horizon EU project LearnData (101086712).

Author contributions

F.B. conceptualized the work. L.B. collected data and performed the analyses. L.G. and J.W. provided methodological insights. F.B. supervised the work. All authors contributed to the interpretation of the results and participated in the writing of the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s41467-025-59031-7>.

Correspondence and requests for materials should be addressed to Federico Battiston.

Peer review information *Nature Communications* thanks Luis Nunes Amaral, and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. A peer review file is available.

Reprints and permissions information is available at <http://www.nature.com/reprints>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025