Unveiling Elite Developers' Activities in Open Source Projects

ZHENDONG WANG, University of California, Irvine
YANG FENG, Nanjing University
YI WANG, Rochester Institute of Technology
JAMES A. JONES and DAVID REDMILES, University of California, Irvine

16

Open source developers, particularly the elite developers who own the administrative privileges for a project, maintain a diverse portfolio of contributing activities. They not only commit source code but also exert significant efforts on other communicative, organizational, and supportive activities. However, almost all prior research focuses on specific activities and fails to analyze elite developers' activities in a comprehensive way. To bridge this gap, we conduct an empirical study with fine-grained event data from 20 large open source projects hosted on GitHub. We investigate elite developers' contributing activities and their impacts on project outcomes. Our analyses reveal three key findings: (1) elite developers participate in a variety of activities, of which technical contributions (e.g., coding) only account for a small proportion; (2) as the project grows, elite developers tend to put more effort into supportive and communicative activities and less effort into coding; and (3) elite developers' efforts in nontechnical activities are negatively correlated with the project's outcomes in terms of productivity and quality in general, except for a positive correlation with the bug fix rate (a quality indicator). These results provide an integrated view of elite developers' activities and can inform an individual's decision making about effort allocation, which could lead to improved project outcomes. The results also provide implications for supporting these elite developers.

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

Additional Key Words and Phrases: Elite developers, developers' activity, project outcomes, productivity, software quality, open source software (OSS)

ACM Reference format:
Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. 2020. Unveiling Elite Developers' Activities in Open Source Projects. *ACM Trans. Softw. Eng. Methodol.* 29, 3, Article 16 (May 2020), 35 pages.
https://doi.org/10.1145/3387111

Z. Wang and Y. Feng contributed equally to this work.
This work is partially supported by the National Science Foundation under awards CCF-1350837 and IIS-1850067.
Authors' addresses: Z. Wang, J. A. Jones, and D. Redmiles, University of California, Irvine, Donald Bren Hall, Irvine, California, 92697-3425; emails: {zhendow, jajones}@uci.edu, redmiles@ics.uci.edu; Y. Feng, Nanjing University, 22 Hankou Road, Gulou District, Nanjing, Jiangsu, China, 210093; email: charles.fengy@gmail.com; Y. Wang (corresponding author), Rochester Institute of Technology, 70, 134 Lomb Memorial Dr. Rochester, New York, 14623-5608; email: oliver.wangyi@gmail.com.

1049-331X/2020/05-ART16 $15.00
https://doi.org/10.1145/3387111

1 INTRODUCTION

Open source software (OSS) has become an engine for innovation and critical infrastructure for modern software development [26]. OSS development is supported by communities formed from a loose collection of individuals. The contribution from these individual developers consists of various software engineering activities, such as coding, bug fixing, bug reporting, testing, and documentation. All of these activities lead to the development and improvement of OSS projects and fundamentally influence their outcomes.

Meanwhile, previous research, e.g., [24, 26, 34, 43, 62], has reported that among hundreds of such individuals, only a small portion of elite developers[1] contribute most of the code and oversee project progress [24, 43, 54]. For example, in Mockus et al.'s study on the Apache community [62], they observed that the top 15 contributors (out of 388 total) had contributed over 83% of modification requests and 66% of problem reports. Beyond committing source code and overseeing projects, elite developers are involved in software engineering activities such as moderating the discussions of an unfixed issue, documenting changes, organizing the project, and communicating with other contributors [34]. Therefore, analyzing the elite developers' activity is critical to understanding the development of OSS projects.

Software developers maintain diverse activity profiles, including implementing new features, documenting changes and design, analyzing requirements, and fixing bugs [54]. Contributing source code is only one type of the activities pursued by an elite developer. Prior studies typically provide insight into one such specific noncoding activity, e.g., peer review [72] or committing code [28]. Most fall short of providing an integrated, holistic view of all of the developers' activities and the distribution of efforts on these activities. These studies provide guidance to software developers on improving some software engineering tasks, such as assigning bug reports [39, 79] and estimating cost [2]; however, we cannot fully realize the activity data to inform better decision-making and ultimately better project output without a comprehensive study of the diverse range of developer activities including their typical technical and nontechnical activities. Because the full range of these activities influences the software systems being developed, holistically understanding the elite developers' diverse activities beyond coding draws the most critical development knowledge and experience from the community. This leads to our first research question:

RQ$_1$ *What do elite developers do in addition to contributing typical technical code in OSS projects?*

Since software engineering is a human-centered activity [36], effectively managing its human resources may significantly enhance project productivity and collaboration quality. However, it is not clear which kinds of tasks they focus on in the development of OSS projects. Understanding the dynamic evolution of elite developers' effort distributions across different activity categories and over the life cycle of OSS projects has many important practical implications. For example, it can guide junior developers on how to adjust their effort distributions during their tenure in a project and can assist in resource management. This gives rise to our second research question:

RQ$_2$ *How does an OSS project's elite developers' effort distribution evolve along with the growth of the project?*

Given that OSS projects are developed by elite developers as well as many external contributors, elite developers' activities, especially those beyond technical contributions, such as communicating with bug reporters, documenting project changes, and assigning tasks and labeling issues, may fundamentally influence the outcome of the entire team. Because successful software engineering

[1]We use the term "elite developers" instead of the more commonly used "core developers" to refer to those who hold clearly defined project management privileges in a project, as opposed to *only* being core *code* contributors.

activities require qualified developers with the proper expertise to complete the task efficiently and effectively, understanding these impacts is critical for developers to oversee the project and ensure productivity and product quality. Thus, we have our third research question:

RQ$_3$ *What is the relationship between an OSS project's elite developers' effort distribution and the project's productivity and quality outcomes?*

To answer the above research questions, we conduct an empirical study using fine-grained event data from 20 large open source projects hosted on GitHub consisting of both company-sponsored and non-company-sponsored projects. To better utilize the activity data to draw insights about the elite developers, we first map them from raw atomic events to sense-making high-level categories. These categories are ***communicative***, ***organizational***, ***supportive***, and ***typical***. Their detailed definitions and mapping protocols are introduced in Section 3.3. We then use multiple techniques to model and analyze the data. Our study reveals three main findings. First, elite developers participate in a variety of activities, in which coding only accounts for a small proportion. Second, with the progress of the project, elite developers tend to be increasingly involved in more nontechnical activities, while decreasing their coding and other technical activities. Third, elite developers' effort distributions exhibit complex relationships with project productivity and quality. For both project productivity indicators (i.e., number of new commits and average bug cycle time in each project-month), our results suggest that project productivity has negative correlations with nontechnical (communicative, organizational, and supportive) activities. For one project quality indicator (number of new bugs in each project-month), our results show that project quality has negative correlations with efforts in nontechnical activities; however, for the other project quality indicator (bug fix rate in each project-month), our results show that project quality has positive correlations with supportive activities.

The main contributions of this article are trifold.

- We conduct an empirical study that not only characterizes elite developers' activities and their dynamics but also identifies the relationships between elite developers' activities and project outcomes. Based on the findings, we identify a set of actionable recommendations for practitioners.
- We take a fresh perspective on investigating the activities of OSS developers through collecting, modeling, and analyzing developers' many kinds of publicly available online software engineering activities rather than focusing on one or several specific activities, thereby obtaining a holistic view of OSS development.
- We set up a *well-cleaned* dataset comprising all the event data of large OSS projects, which is made publicly available.[2]

Our work is built on SE communities' continuous efforts in investigating and assisting OSS projects in the last two decades. Researchers have investigated community structures and compositions, individual motivation, behavior and experiences, and these factors' impacts, e.g., [9, 44, 67, 69, 85, 87, 89]. While these extant studies build solid knowledge on OSS projects, most of them focus on the code-contribution-related activity, such as coding, reviewing, testing, debugging, and so on. Our work expands the literature by enhancing understandings about the breadth and dynamics of elite developers' activities and their correlation with project outcomes.

The rest of the article proceeds as follows. Section 2 briefly reviews the background and related work. Section 3 introduces the research design. Section 4 reports results and findings. Section 5

[2] https://drive.google.com/drive/folders/10ibmz2svPRf3jfRtm7mbiouo9ATaYAoB.

discusses findings, implications, and threats to validity. Section 6 concludes the whole article and outlines potential future work.

2 BACKGROUND AND RELATED WORK

In this section, we briefly overview the backgrounds for this study and discuss the related work about open source communities and developers' activities. We start from a brief introduction of the hierarchical structure of open source communities, followed by discussions of relationships between developers and the activities/workload of developers. We also highlight how the current work distinguishes itself from prior work.

2.1 The Hierarchical Open Source Community

Open source development has been a mainstream practice in building modern software systems [26]. Different from traditional development paradigms, an open source project is centralized in its community, which produces collective goods through collaboration among its members [41]. Though the detailed governance and social practices may vary across different projects [65], members of an open source community usually have different roles, which entail different responsibilities, rights, and levels of contributions [10, 76]. Similar to other hierarchical organizations, an OSS community follows an onion-shaped social structure [24].

There are several different definitions for each layer in this hierarchical community [24, 34, 43], but in general there are five major types, radiating outward: from core developers, to the internal and external contributors, to issue reporters, and finally at the outermost layer to peripheral members (note that terms may differ from study to study). However, each member in a project may play several roles.

Peripheral members of an OSS project typically are users of the software artifacts who do not directly contribute to the project other than sending user feedback or usage data. For most users of an OSS project, a peripheral member is the starting point, unless he or she has achieved recognition in the same ecosystem [44]. If these peripheral members wish to contribute to more critical tasks of the project, they usually have to undergo a socialization process. In Ducheneaut's case study [34], he reveals the socialization path of becoming a core developer when starting at the periphery. This path includes socialization with the current core team and completing a series of development tasks that increase in difficulty. After being socially recognized by experts for a project, they join the core team to become core developers and gain the privileges of this project (i.e., get project "tenure" in a repository). Further, they begin to hold administrative power in the project; for example, they can oversee other external contributors' technical submissions.

Current OSS development, especially large-scale projects, can be described under the "umbrella" of an ecosystem. In a follow-up study, Jergensen et al. discussed the evolution of this socialization process in the context of modern OSS development [44]. As software engineering technologies continue to develop, such as advances in version-control systems (*git*), fewer open source projects are being developed in isolation. Further, more projects are developed in parallel under the broader context of software ecosystems. In their study, they found that there are several types of contributors among open source users across different projects [42]. In addition, many developers move from project to project like "nomads." Another critical finding is that as developers in an OSS project gain more technical experience, their contribution is not toward the core of the project in terms of code centrality.

Among many studies on OSS communities, researchers have come to the consensus that only a small portion of developers make most of the contributions [24, 43, 54]. Understanding elite developers is critical in investigating the health and sustainability of the community, and various methods have been employed to analyze their activities. In most mature open source projects,

Table 1. Comparisons between Elite Developers and Other Similar Roles Defined in Literature

Role	Complement Role	Scope	Role Definition
Core	Peripheral	Open source	A small group of members who are mainly responsible for overseeing and contributing to the project [24, 26, 62]
Maintainer	Contributor	Open source	Members who are responsible for a software module, mainly in accepting contributed patches [34, 43, 102]
Internal	External	Company-sponsored open source	Individuals who are members of the development group; usually are listed as contributors on the project homepage [32, 92]
Mentor	Newcomer	General software development	Persons who train and help novice and inexperienced developers (newcomers) for their onboarding [5, 17, 30]
Elite	Nonelite	Open source	Developers who own administrative privileges in the project [this study]

members have developed a shared basis of authorities and privileges ("bazaar" in Eric Raymond's ideology [71]), transferring authorities and privileges among them. Thus, a member's identity as "an elite" is indeed dynamic [65].

2.2 The Role-Based Relationships among Developers

Members of a software project form complex relationships with each other while performing a wide range of development activities. Table 1 shows a brief overview of studies investigating the relationships between developers. While "core vs. peripheral" is a dominant terminology used in SE literature to characterize role-based developers' relationships, other terminologies have been proposed. For instance, when outside or novice developers are in the process of joining and learning from an existing project, the *mentor* and *newcomer* relationship between developers would eventually be established for social and technical reasons [17, 30]. Prior research also suggests that mentorship activities face several barriers for both ends of stakeholders [5], which threatens project sustainability. In another scenario, when software companies open source their internal software to the public domain, understanding the contribution behaviors and impacts of *external* and *internal* contributors for these company-sponsored projects becomes critical for many reasons. For example, companies might want to find a balance of management efforts and fast iteration of enhancement when receiving external help. As mentioned earlier, developers' roles change over time, and role migrations are very common [44]. A peripheral member could be promoted to a core member, and a newcomer could be a mentor after his or her skills have developed. However, such a growing process may take substantial effort. For instance, external members may need to undergo a time-consuming process to gain the roles of internal members [32, 34]. Besides, role migrations are not unidirectional. For example, core members may lose their roles if they no longer actively contribute to the project [58]. Therefore, role-based relationships are dynamic in nature.

Open source software and its developer community have been substantially evolving since the 2000s. The transparency afforded by online open source code hosting sites such as GitHub enables researchers and practitioners to closely observe and review the dynamics of the projects, such as the evolution of software artifacts and the trajectories of peripheral participants' self-development

[29, 34]. Meanwhile, supported by version-control systems and logs from various communication channels, a tremendous volume of social and technical latent data can be acquired [8]. Various empirical research has been postulated to obtain valuable knowledge from these datasets for software design, development, and quality assurance. Besides, activity data are critical in identifying role-based relationships among developers.

To extract insights from an open source project's socio-technical processes, and thereby improve them, information about role-based relationships is crucial [45]. Such information could be employed to extract developers' expertise [6, 63], model a project's life cycle [46], or predict project outcomes [18, 90]. Realizing these utilities require identifying role-based relationships from multiple perspectives, researchers often rely on count-based measurement for technical contributions. Such work is based on a simple assumption that "core" developers perform more activities than others Crowston et al. [27] and Mockus et al. [62]. However, this perspective may be not sufficient for all scenarios. Recently, Joblin et al. compared network-based perspectives for identifying core and peripheral developers, which captures structural roles beyond counting technical contributions. It shows that researchers may need to take fresh perspectives in recognizing developers' roles to fit their own research scenarios.

2.3 The Activities and Workload of Developers

A number of studies have aimed to investigate developers' activities and tasks in software engineering research, including [24, 34, 102], providing critical insights on how project members collectively deliver software systems in a context involving many factors. For example, Baltes and Diehl [6] developed a conceptual theory to characterize complex relationships among developers' activities, the formation of expertise, and performance by surveying 335 software developers and literature on expertise and expert performance.

There have been research efforts attempting to categorize diverse activities to a few sense-making high-level categories to derive actionable insights for software development. In an early study, organizational psychologist Sonnentag conducted an empirical study with software company professionals to study their weekly activities in software development [80]. Based on her observations and the grounded theory process, she classified four broad types of activities in the professional lives of developers: *communicative*, *organizational*, *supportive*, and *typical*. According to her field study with excellent and average software professionals, she found that both types of developers usually spend a similar amount of effort on typical software engineering tasks such as coding, testing, and debugging, while excellent developers spend more time meeting and consulting. This study is critical in identifying and classifying the full spectrum of software engineers' activities, which serves theoretical foundations for the analysis in our work. However, we still need to develop a mapping between fine-grained raw events in open source development and these high-level categories (see Section 3.3).

Performing these technical and nontechnical activities is not effortless. Thus, studying developers' workload becomes necessary. In an empirical study with the Linux kernel ecosystem, Zhou et al. [102] highlighted that the workload per maintainer varies according to modules, which forms different equilibria to make these modules sustainable. However, the workload per maintainer tends to be stable along with the expansion of the ecosystem. Their results further revealed that the workload among developers is highly unbalanced; i.e., a small number of developers often have to bear a lot of workload increase. To reduce such a workload imbalance among Linux kernel maintainers, researchers have proposed alternative workflows, for example, the multiple-committer model in [84].

Roles in open source project also determine members' different activity focuses and work efforts [51]. Nevertheless, in SE literature, the complex division of labor [7] in an open source

Table 2. Sampled Projects and Their Description

Project	Description
Aframe	Web framework for virtual reality applications
Alamofire	Swift library for HTTP networking
ExoPlayer*	Media player for Android
Finagle*	Extensible RPC system for JVM
Fresco*	Android library for images
Guava*	Set of various Java libraries
Immutable-js*	JavaScript library for immutable data structure
Jest*	JavaScript testing framework
Marko*	JavaScript library for building UI
Moya	Swift network framework
Nightmare*	Browser automation library
Rclone	Program to sync files
React*	JavaScript library for building UI
Recharts	JavaScript chart library
Sqlitebrowser	Visual UI for databases in SQLite
Stf	Smart device testing framework
Tensorflow*	Library for numerical computation
Tesseract	Text recognition (OCR) engine
Tidb*	Distributed database system
ZeroNet	Decentralizes websites to be resistant to censorship

*Projects sponsored by companies.

project has not been well studied, particularly, with explicitly considering role-based relationships and different types of activities beyond technical contributions. For instance, Zhou et al.'s study [102] focused on maintaining source code files, which restricts its results to technical work only. Their work also did not distinguish different types of maintainers. Our "elite vs. non-elite" terminology is exactly designed to capture such an often-neglected perspective, since the *elite* obtain the managerial responsibilities in an open source project. Such a perspective, combined with recognizing different types of activities including both technical contributions and nontechnical efforts, can offer valuable insights regarding developer roles that may not be captured by other existing perspectives and corresponding operationalizations.

3 EMPIRICAL STUDY DESIGN

To answer the three research questions presented in Section 1, we conduct an empirical study based on 20 open source projects. This section introduces the design of the study.

3.1 Targeted Projects

We selected 20 open source projects hosting their repositories on GitHub as the targets of the study. Table 2 lists them with short descriptions. The selection of the targeted projects is based on four considerations. First, the selected projects are all projects with established administration structures (i.e., they have a formal project management committee and solicit contributions through the pull-request model) and must be large enough and have traceable records of continuous contributions from a set of contributors (at least 100 pull-requests and 50 contributors historically). Second, the selected projects represent a diverse sample of projects in terms of application domains, such as a testing framework (jest), a popular deep-learning library (Tensorflow),

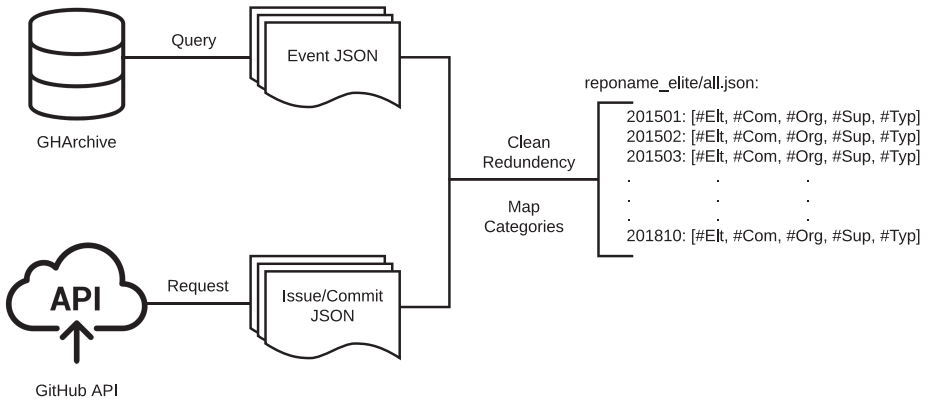


Fig. 1. Data collection and cleanup process.

a multimedia player (ExoPlayer), a web development framework (React), and a database (Tidb). Third, our sample includes a subset of company-sponsored ($n = 11$) projects, which reflects the trend of the increasing involvement of companies in open source development [92]. Last, the sampled projects should maintain relatively long traceable records on GitHub, which allows us to study the longitudinal dynamics while maintaining data consistency.

3.2 Data Preparations

The current version of the GitHub API only allows us to retrieve 300 events or events from up to the past 90 days, whichever is met first.[3] Therefore, in order to extract event data from an extended range of projects' life cycle, we employ the GitHubArchive public data dump on Google Cloud. We also employ Google BigQuery to extract the monthly event log for each sampled repository from January 2015 to October 2018. Additionally, for repositories that started or were made public during the year 2015, we store data files starting from the project creation month. Figure 1 provides an overview of the data collection and cleaning process.

For each month, GHArchive provides most of the activity events such as push, open issue, open pull-request, Gollum (editing wiki), and comment for project repositories. We use SQL-like queries (designed by BigQuery) to search for projects and save the results in tables of the personal Google Cloud database. Further, we export tables as JSON files to the cloud storage and download them to a local computer for later analyses.

In total, we collected 5.60GB of 900,862 events (communicative: 238,986; organizational: 42,317; supportive: 514,957; and typical: 104,602) for these 20 repositories.

However, several types of events associated with issues could not be recorded using the above method, e.g., assigning a "won't fix" label to an issue by a project administrator or delegating a developer to investigate a newly posted bug. To fix this problem, we resort to a customized Python script via the request[4] library to request events from the GitHub API and then to download issue event logs for every issue that has been reported in each repository. Thus, we collect precise project management information, such as who has the administrative privilege on a repository and oversees the progress of the project. To more easily search for commits by date and author and derive necessary metrics for the later data analysis on the project productivity, we also download

[3] GitHub Event API: https://developer.github.com/v3/activity/events/.
[4] Simplified HTTP request client for Python: https://github.com/request/request.

the commit logs of all sampled projects. In total, we have collected 1.81GB data of issue events and commit logs.

Finally, we use Python scripts to merge event data based on event ID and commit SHA and clean the redundant data that were recorded on both data sources. By using two data sources, there are some categories of events that were kept recording on each data source, such as *close issue* and *reopen issue*. Because the GHArchive project employs the GitHub event API to archive activities on a daily basis, we decided to keep events from the GitHub Issue API. We convert event logs into a monthly list based on the number of events that occurred in each major category.

3.3 Event Categories and Mapping

Although the event log data faithfully records developers' activities, we need to recode the data unit categories so that they are easier for humans to understand and analyze. Particularly, the percentage of types for a developer group should be able to reflect the effort allocation and focused roles.

Collecting low-level activities from self-reported and observed data in the field, inductively mapping these activities onto broad categories, and systematically extracting behavior patterns and analyzing work effort allocation is common practice to establish the activity profile of a certain group [16, 80, 81, 100]. One prior field interview study [16] focuses on the daily activities category of professional software developers, which were summarized based on subjects' reported activities. Because we are particularly interested in investigating the overall activity profile of elite developers, we choose to follow an established category system that reflects the daily activity, instead of another low-level task-based category system that focuses on coding activities.

We reuse the categorizing system created by Sonnentag [80] to further investigate the contribution of elite developers, as well as the relationships between their effort distribution and project outcomes for open source projects. This study summarizes and categorizes professional software developers' daily activities into four major categories: *communication*, *organization*, *support*, and *typical*. In their study, research subjects were developers in private companies; to adapt these definitions to open source development, we slightly modify the definition and operationalization of each category.

3.3.1 Communicative. In the conventional colocated software development team, communicative activities usually refer to formal and informal meetings and consultations [80]. However, under the setting of distributed software development, which open source projects usually employ, each project applies various communication channels, including mailing lists, instant messaging, and online discussion boards [8]. Moreover, some projects such as Tensorflow apply additional broadcast channels, such as a blog, website, and/or YouTube channel. Therefore, similar to other empirical studies with open source developers, we are not able to collect communicative activities on all channels. For example, private instant messages are often unavailable. However, as GitHub is the major platform for developers to exchange ideas, by extracting communicative event logs from GitHub, we can capture all *public communicative traces* that happened on this platform by each contributor.

The definition of communicative activities is *public and visible communication through commenting features supported by the platform on issues, commit, and project milestones.*

3.3.2 Organizational. In previous field studies, organizational activities are categorized as delegating tasks among the development team and other project organizations in professional software development. Similarly, under the open source development settings, representative activities of this type are *assigning* and *unassigning* tasks to a developer, such as assigning someone with a GitHub issue or reviewing a pull request.

We define organizational activities as *managing the project community and delegating tasks, including code reviewing, debugging, and user support to internal and external contributors through the features supported by the platform.*

3.3.3 Supportive. Supportive activities are critical to open source development and mainly refer to other noncoding activities in collaboration. It includes *documentation* work such as writing documentation/wikipage and categorizing issues by adding labels to them. Further, supportive also includes *maintenance* work, for example, managing development branches and releasing or archiving code versions.

We define supportive activities as *noncoding activities in the collaborative open source development that are performed through techniques supported by the platform, including documentation, versioning control, and development branch management.*

3.3.4 Typical. Typical activities in software development are coding, testing, debugging, and reviewing on an *individual* basis. Thus, under the setting of the open source platform, we only include commit activity under this category. In addition, we count event actor as the commit *author* rather than the *committer*, since the *author* is the original developer who wrote the code.

We define typical activities as *conventional code-writing tasks finished at the individual level and counted as submitted code reviews, commits, and pull requests.*

3.3.5 Mapping Raw Events to the Above Categories. We apply the closed card sorting method to place 35 raw GitHub events into the above four major categories. Three researchers perform the card sorting. Among three researchers, the card sorting yields 0.92 average joint probability of agreement; it achieves 82.8 % relative observed agreement and reaches 0.77 kappa [15]. All these metrics indicate good interrater reliability, and all differences among card sorters are discussed and resolved. The final mapping is shown in Figure 2.

In the aforementioned card sorting and disagreement resolving process, we also consider the context of a specific type of event in making classification decisions rather than blindly relying on its literal name. Let us take events related to "comment" as an example. There are three major events on GitHub that contain the keyword comment in their event names: *CommitComment*, *IssueComment*, and *PullRequestReviewComment*. The first two are quite straightforward. According to their definitions on GitHub, *CommitComment* and *IssueComment* are almost always used for communication purposes,[5] and thus we classify them as communicative activities. They almost never lead to direct changes to a project's code base. However, the event *PullRequestReviewComment* is not that straightforward. Commenting on a *PullRequestReview* seems to be for communication purposes; however, it directly relates to technical contributions and determines whether such contributions should be accepted. In GitHub's pull-request model, a pull-request initiates a code review process. In such a process, a *PullRequestReviewComment* refers to *submitting a comment on a pull -request's unified diff* (review to a specific `diff`, see the comment B in Figure 3 as an example). It often directly results in merging or closing the pull-request to which the comment pertains. Thus, similar to its closely related event *PullRequestReview* (see the review A in Figure 3 as an example), it shall be classified as a kind of typical technical work.

By mapping the low-level raw GitHub events into these four categories, we can reason about developers' activities at the level that makes sense to understand them as real work practices and human efforts in organizational settings [66], instead of losing in millions of atomic events that are not considered as integrated work practices. We argue that such a categorization system precisely

[5]According to Chris Wanstrath, founder of GitHub, these two "commenting" features were designed to replace the email chains for communicating commit/issue, see https://github.blog/2008-04-10-commit-comments/.

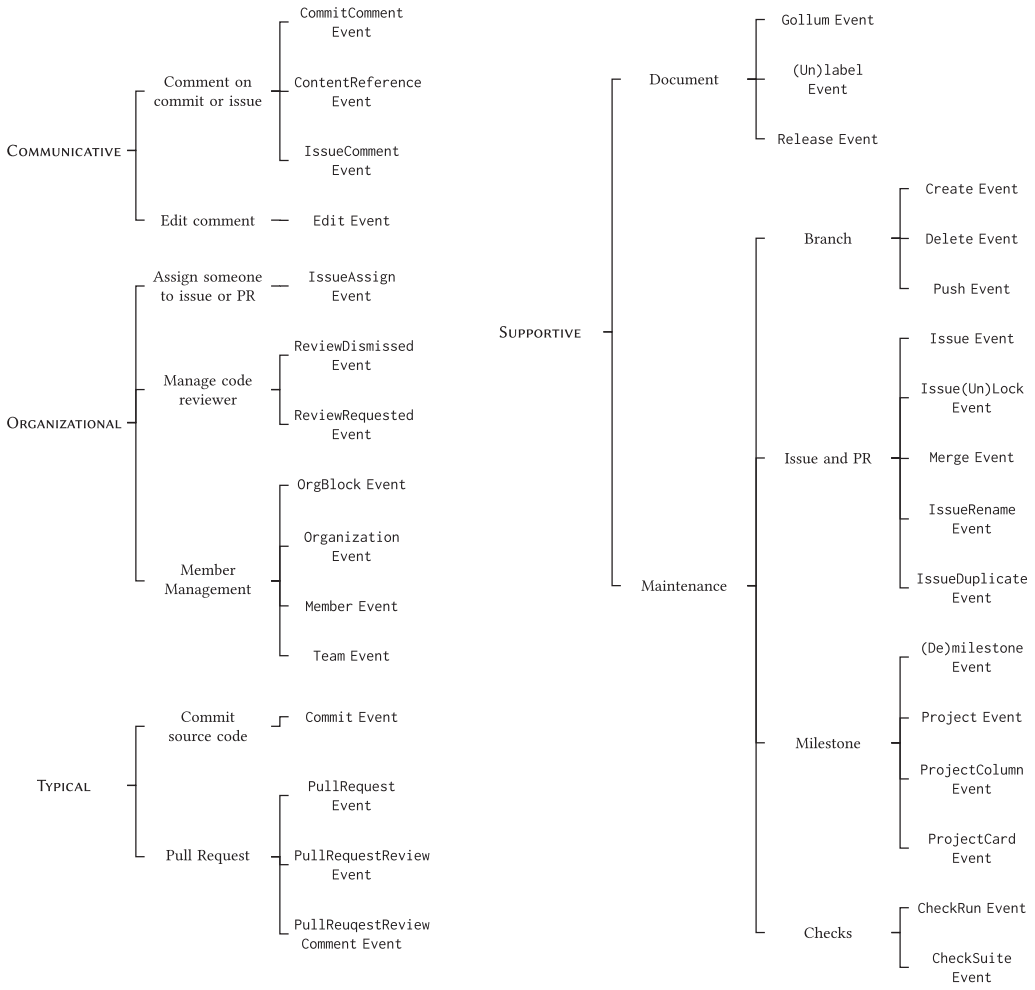


Fig. 2. The taxonomy of GitHub event types. The definition of each raw event can be found in the official GitHub Events API documentation page: https://developer.github.com/v3/activity/events/.

and comprehensively reflects the general work practices of professional software developers. The categorization system is borrowed from the literature [16, 80] of empirical field observations and interviews with a large number of software development projects and hundreds of professional developers. Although our study focuses on open source developers, the types of their routine work practices at the individual level in software development would be unlikely to go beyond the in-house software development, while the way of organizing such practices may be different at the collective level [23, 38, 74, 91]. Doing so enables us to better study the dynamics of elite developers' work practices and their impacts, thus deriving meaningful findings and implications.

3.4 Collecting Project Outcomes Data: Productivity and Quality

Since one of our research goals is to investigate the impact of elite developers' activity on project outcomes (RQ_3), we need to collect project outcomes data. We consider two project outcomes:

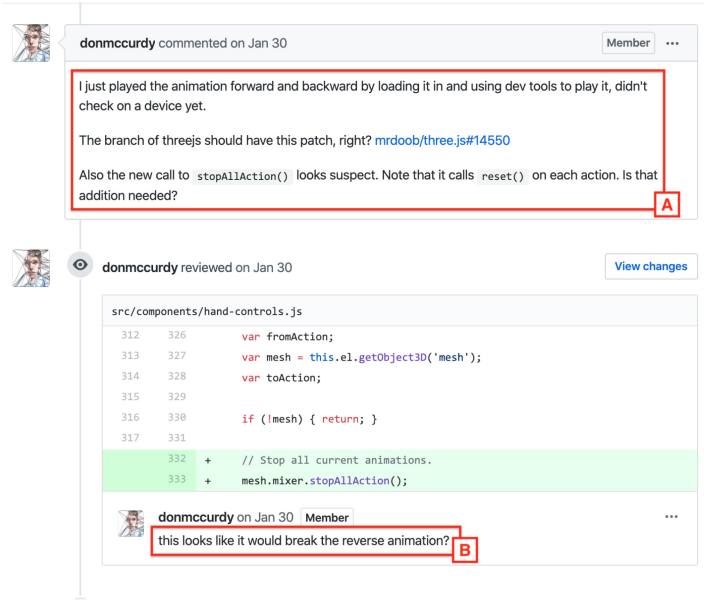


Fig. 3. The overall code review in highlighted part A triggers *PullRequestReview* and specifically to a unified diff in part B, which triggers*PullRequestReviewComment* Event.

productivity and quality, which are viewed as the most important [90]. Each of them has two indicators, which are introduced as follows.

For productivity, the first indicator is *the number of a project's new commits in a project-month.*[6] Thus, for project i in month m, we use $NewC_{im}$ to denote it. In many studies focusing on the OSS development and community, the number of commits is considered as the productivity metric (e.g., [88–90]). We adopt this widely used productivity indicator. Note that we count the commits from all contributors rather than from elite developers only, because we measure the impact on the productivity of the whole team. The second indicator is *the average cycle time of a project's closed bugs in a project-month.* Similarly, for project i in month m, we use BCT_{im} to denote it. Such an indicator has been used to measure project productivity in many prior studies (e.g., [50]).

Following the conventions in previous SE literature (e.g., [48, 70, 90]), we first operationalize the code quality by *the number of bugs found during a project-month.* We simply use $NewB_{im}$ to denote it. On GitHub, the issue can be of various types, e.g., discussion, new feature request, improvement request, and so on. To categorize these issues, software developers often employ some keywords to tag them. However, because tagging is often project specific, we adopt Vasilescu et al.'s [90] method to distinguish bug issues from other issue types in this study. We set up a list of bug-related keywords, including *defect, error, bug, issue, mistake, incorrect, fault, and flaw*, and then search for these words in both the issue tags and issue titles. If any tags or title of an issue contains at least one keyword, we identify it as a bug issue. Similarly, as the productivity data, we compute the number of new bug issues in every project-month. In addition to counting the newly found bugs in each project-month, our study also includes a second quality indicator: Monthly Bug Fix Rate

[6]To simply the following discussion, we use the term "project-month" to denote *a given month in a project.*

(BFR_{im}), which is defined as

$$BFR_{im} = \frac{No.\,of\,Fixed\,Bugs}{No.\,of\,Found\,Bugs},\ for\,project\,i\,in\,month\,m.$$

The Bug Fix Rate is one of the key metrics related to the defect removal process [33, 95]. In fact, it partially represents the effectiveness of the quality assurance process by characterizing the birth (finding a new bug) to death (fixing a bug) process of defect removal [56]. If $BFR_{im} < 1$, it indicates that the project's quality risk is accumulating.

3.5 Identifying the Elite Developers

Following the method used in Hanisch et al.'s study [40], we leverage GitHub's repository permission mechanism to identify elite developers. Attaining the status of elite developer in a project means that a developer has obtained write permission for an organization's repository. By gaining this level of permission, the developer can perform many tasks on a repository without requesting, for example: directly pushing commits to a repository; creating and editing releases; and merging pull-requests. In addition, with write permission of the repository, the developer can perform several types of administrative work, such as submitting code reviews that affect a pull-request's mergeability, applying labels to tasks and milestones to the repository, and marking an issue as a duplicate, which allows the issue to lose public attention.

Unfortunately, GitHub does not allow anyone other than the repository owners to access the list of members obtaining specific permissions. We apply a permissions check mechanism to determine the elites. When a developer in the repository performs a task that requires write permission, we tag this developer with "elite-ship" of the repository. As we observed in this study, we found that a project's elite developers might also suffer a survival issue [58], and thus we set 90 days[7] as the length of the "elite-ship" and use this time window to filter developers who were inactive. During this 3-month period, if this developer performs any task that also requires write permission, his or her "elite-ship" would be renewed for another 3 months, starting from the month when he or she performed the task.

Compared with other elite developer identification methods based on metrics or networks [45], our methods have several advantages. First, our method takes a dynamic view of the elite developer status in the open source community where developers have very high mobility in terms of entering and leaving.[8] Second, our method reflects the socialization process of gaining power and status in a community. Third, our method respects the fact that some developers may be nominated as elite developers before making substantial contributions, particularly in the company-sponsored projects. Lastly, our method avoids dealing with the marginal cases resulting from the arbitrarily set threshold, e.g., the one-third cut-off used in [27].

3.6 Data Analysis

Table 3 presents the mapping between **RQs** and corresponding data analysis methods. We will introduce them in detail in the rest of this section.

For $\mathbf{RQ_1}$ and $\mathbf{RQ_2}$, we only need to apply simple statistical methods to answer them. Answering $\mathbf{RQ_3}$ requires one to establish correlations between effort allocations and project outcomes. We use panel regressions, which is a type of econometric techniques, to build the linear models. Because our data is panel data in its nature, cross-sectional (multiple projects), longitudinal (multiple

[7]Literature on survival analysis of open source developer usually use 30 days or 90 days as a time window [58]. When we examined the raw data we have, we found that 30 days (1 month) is too short, while 90 days (3 months) is a moderate time interval to avoid rush decisions on judging whether someone had gained/lost the elite identity.

[8]For company-sponsored projects, the mobility may also result from organizational and individual career changes.

Table 3. Research Questions and Corresponding Data Analysis Methods

RQs	Data Analysis Methods
$\mathbf{RQ_1}$	Descriptive statistics
$\mathbf{RQ_2}$	Descriptive statistics, ANOVA
$\mathbf{RQ_3}$	Project-specific fixed effects Panel Regressions (LSDV estimator with Diagnostics)

time windows for a specific project), ordinary multivariate regressions such as OLS cannot deal with such types of data [99]. Moreover, panel regressions provide another benefit. They already deal with commercial, social, and technical confounding factors implicitly and inherently. For example, project-specific factors (e.g., project domain, etc.) are treated as unobserved time invariant in regressions. Therefore, the independent variables' effects could be precisely estimated without worrying about potential interactions and selective biases[9] with many confounding factors when we do not aim for causality (see Wooldridge [99]: Chapter 7, p. 256; Chapter 14).

All statistical analyses are performed with R 3.4.1 [68] and its associated packages for macOS High Sierra (version 10.13.1). We follow the ASA's principles to present and interpret statistical significance [94].

3.6.1 Summarize Activities for $\mathbf{RQ_1}$. Answering $\mathbf{RQ_1}$ does not require complicated analysis techniques. We use descriptive statistics to derive results and findings for this research question. Note that we code the raw GitHub activities into four broad activity categories (communicative, organizational, supportive, and typical) according to [80] (described in Section 3.2). Doing so helps us to derive meaningful insights instead of fragile, overly detailed information in the raw activities. For all sampled projects, we calculate the total of elite developers' activities over the four broad categories. Thus, we have a 4-tuple for each project as follows:

$$< Com, Org, Sup, Typ > .$$

We also compute the percentage of elite developers' activities over the entire project's activities. All results are reported in Section 4.1.

3.6.2 Identifying Activity Trend for $\mathbf{RQ_2}$. To answer $\mathbf{RQ_2}$, we first group the activities according to the month of their occurrences. Then, similarly, for a project i in each month m, we can calculate a similar 4-tuple:

$$< Com_{im}, Org_{im}, Sup_{im}, Typ_{im} >,$$

where $i \in \{1, \ldots, i, \ldots, 20\}$, and $m \in \{1, \ldots, m, \ldots, 36\}$.

Since the different projects have different numbers of elite developers, cross-project comparisons require us to average the project-level data to the individual level. We simply calculate the average activities per developer over the four categories. Then, we can calculate the individualized monthly growth rates of activities in each category for each project. Given that there are 20 projects, for each category we have 20 growth rates. We use one-way ANOVA to see if there is any difference across the four categories regarding the growth rates.

[9]When there are many commercial, social, and technical confounding factors that cannot be enumerated in statistic analyses, selecting a subset of them would result in selective biases.

3.6.3 Identifying Correlations with Project Outcomes for $\mathbf{RQ_3}$. Answering $\mathbf{RQ_1}$ and $\mathbf{RQ_2}$ provides the data we need to answer $\mathbf{RQ_3}$. Before discussing the analysis methods, we first examine the data.

We want to investigate the correlations between a project's elite developers' effort distributions and project outcomes. The *independent variables* are the effort distributions over the four categories of activities, which can be easily extracted from the collected data. The dependent variables are four indicators of project outcomes (productivity: $NewC_{im}, BCT_{im}$; quality: $NewB_{im}, BFR_{im}$), which are adapted from the prior software engineering literature. Given that we have broken a project's data into months when answering $\mathbf{RQ_2}$, and using "month" as the analysis unit, we have one data case for each project i at each month m. Therefore, we have 720 (20 projects × 36 months) data cases, in total. Each data case is in the following form:

$$< NewC_{im},\ BCT_{im},\ NewB_{im},\ BFR_{im},\ \overline{S-Com_{im}},\ \overline{S-Org_{im}},\ \overline{S-Sup_{im}},\ \overline{S-Typ_{im}} >,$$

where $i \in \{1, \ldots, i, \ldots, 20\}$, and $m \in \{1, \ldots, m, \ldots, 36\}$.

The $\overline{S-Com_{im}}$ represents the share of communicative activities in all four categories of activities per elite developer for project i in month m. Similar denotations apply to the other three. Note that

$$\overline{S-Com_{im}} + \overline{S-Org_{im}} + \overline{S-Sup_{im}} + \overline{S-Typ_{im}} = 1. \tag{1}$$

Answering $\mathbf{RQ_3}$ is identifying the relationships between these four independent variables and four dependent variables $NewC_{im}$, $BugC_{im}$, $NewB_{im}$, and BFR_{im}. A natural solution is performing regression analysis. Our data is panel data (cross-sectional: from 20 projects; longitudinal: 36 months per project). Thus, simple OLS multivariate linear regression is not a proper technique because we cannot assume there is no difference among the 20 projects and 36 data points.

As we mentioned before, to correctly identify the relationships, we employ panel regression methods to deal with the panel data [99]. Intuitively, each project has its own characteristics, so we use the project-specific fixed effects models.[10] The analyses estimate parameters for the following four regression Equations (2) to (5):

$$NewC_{im} = \beta_1 \times \overline{S-Com_{im}} + \beta_2 \times \overline{S-Org_{im}} + \beta_3 \times \overline{S-Sup_{im}} + \alpha_i + u_{it} \tag{2}$$

$$BCT_{im} = \beta_1 \times \overline{S-Com_{im}} + \beta_2 \times \overline{S-Org_{im}} + \beta_3 \times \overline{S-Sup_{im}} + \alpha_i + u_{it} \tag{3}$$

$$NewB_{im} = \beta_1 \times \overline{S-Com_{im}} + \beta_2 \times \overline{S-Org_{im}} + \beta_3 \times \overline{S-Sup_{im}} + \alpha_i + u_{it} \tag{4}$$

$$BFR_{im} = \beta_1 \times \overline{S-Com_{im}} + \beta_2 \times \overline{S-Org_{im}} + \beta_3 \times \overline{S-Sup_{im}} + \alpha_i + u_{it}. \tag{5}$$

Note that we do not include $\overline{S-Typ_{im}}$ into Regression Equations (2) through (5). The reason is straightforward: the sum of $\overline{S-Typ_{im}}$ and the other three is always "1" according to Equation (1). Thus, it is perfectly correlated with the other three. Including it will lead to a significant multicollinearity problem.[11]

For each dependent variable, we use the least-squares dummy variable (LSDV) estimator to estimate the parameters in the project-specific fixed effects models. After we finish the model estimation, we perform a series of regression diagnostics for examining the time-specific effects and empirically justifying the use of fixed effects models. These regression diagnostics include time-fixed effects testing, F-test for (`pFtest`), Hausman Test (`pHtest`), Heteroskedasticity testing,

[10] We also empirically perform model diagnostics, which prove that fixed effect models are better than both OLS and random effects models; see Section 4.3.1.

[11] In fact, no coefficient can be estimated for it in R.

Table 4. Activity Amount That Has Happened on Each Sampled Project

Project	Com.		Org.		Sup.		Typ.	
	Total	Elite%	Total	Elite%	Total	Elite%	Total	Elite%
Aframe	4908	0.46	455	0.99	19400	0.85	5180	0.72
Alamofire	5967	0.18	1773	1.00	11906	0.61	1465	0.62
Exoplayer	9293	0.32	2293	1.00	22197	0.71	5361	0.87
finagle	2488	0.30	46	0.93	2947	0.46	2753	0.49
fresco	5283	0.29	290	1.00	10481	0.64	1923	0.77
guava	3161	0.29	664	0.99	7724	0.71	2239	0.53
immutable-js	2909	0.15	28	0.75	5869	0.59	1057	0.52
jest	19073	0.36	1025	0.99	39995	0.66	5015	0.43
marko	1937	0.38	403	0.95	5525	0.79	2956	0.93
Moya	5376	0.43	416	0.61	22808	0.42	2860	0.73
nightmare	3105	0.14	24	1.00	4963	0.48	892	0.50
rclone	7475	0.23	182	1.00	15243	0.66	2781	0.80
react	35086	0.37	3730	1.00	83036	0.74	9640	0.59
recharts	3199	0.15	54	0.85	4980	0.41	1396	0.66
splitebrowser	6129	0.42	493	1.00	11589	0.70	1751	0.84
stf	1694	0.33	25	0.64	2672	0.55	837	0.69
tensorflow	97940	0.48	28236	0.92	183485	0.75	43029	0.50
tesseract	4870	0.35	116	1.00	9805	0.59	2512	0.55
tidb	16240	0.80	1944	0.98	45451	0.91	8305	0.89
ZeroNet	2853	0.27	120	1.00	4881	0.56	2650	0.79
Mean	**11949.30**	0.34	**2115.85**	0.93	**25747.85**	0.64	**5230.10**	0.67

and so on. Given that our sampled projects consist of 11 company-sponsored projects and 9 non-company-sponsored ones, it is natural to investigate if effort distributions' impacts on project outcomes are sensitive to these project characteristics. Therefore, we perform the same regression analyses to the two subsamples. The results are reported accordingly. All the panel regressions, if not otherwise stated, are performed with R's `plm` package [22].

4 RESULTS AND FINDINGS

In this section, we report the results and findings. We organize them according to the three **RQ**s. All data has been made publicly available for download.[12]

4.1 RQ_1: Elite Developers' Activities

Table 4 provides the basic descriptive statistics of the activities in each project according to their categories. Except for the communicative activities, elite developers perform over 50% of the activities for those in all three of the remaining categories. For each project, our results have confirmed the finding from other studies on the core or elite developers of open source communities, e.g., [34, 62, 93], and elite developers in the community contributed most of the source code submission. In our sample, 67% of typical development tasks are performed by a project's elite developers.

[12] All experiment results, including intermediate outputs and raw data, can be downloaded at https://drive.google.com/drive/folders/10ibmz2svPRf3jfRtm7mbiouo9ATaYAoB.

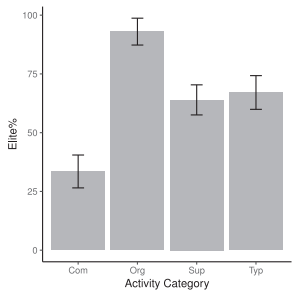


Fig. 4. Elite developers' effort allocations in each category.

In addition to elites' code submission, we also found empirical evidence that elite developers are also responsible for most other types of events. Besides organizational events (according to our definitions, most organizational events automatically require the write permission), elite developers performed over 60% of supportive activities and even created 34% of communicative activities. See Figure 4 for the percentage distribution of elites' contribution.

Moreover, the average numbers of activities performed by an elite developer in a project-month are much higher in all categories when compared to a nonelite developer (see Figure 5). We observe orders of magnitude differences between them. On average, an elite developer performs 7 times more communication activities, 145 times more organizational activities, 22 times more supportive activities, and 22 times more typical activities than a nonelite developer per month. Thus, on an individual basis, we argue that elite developers may have major impacts on projects based on their activity amount.

Answers to RQ_1. Based on the events in each category, we can answer RQ_1 as follows:

On GitHub*, elite developers have contributed to the project in various ways in addition to performing over 60% of the code contributions. They need to manage the community by delegating tasks to other developers with special expertise, managing parallel development among contributors, creating documentations for the project, and participating in discussions with teammates, external developers, and peripheral users.*

4.2 RQ_2: The Evolution of Elite Developers' Activities

4.2.1 Individual Activities of Elite. We find increasing trends in communicative, supportive, and organizational events for each elite developer in our sample. Take the most complex project in our project sample, *Tensorflow*, as an example. The red lines in Figure 6 describe the changes of average activities of this project's elite developers. Though supportive events change dramatically because of the period of software patches and releases, it still exhibits a growing pattern in the long run. The increase of organizational events may be due to the scale increase of the team (the number of active elite developers has increased from 29 to 270 for *Tensorflow*). However, we found the amounts of typical technical activities by elite developers (on average) are stable after the initial project release phase, even for fast-growing projects such as *Tensorflow*.

In contrast, such increasing trends do not exist for the nonelite. For example, the blue lines in Figure 6 represent average activities' changing trends for *Tensorflow*'s nonelite developers. It is easy to find some decreasing trends on communicative and supportive activities, which display opposite trends compared to the elite developers' averages. Additionally, the trend of nonelite developers' typical technical activities is similar (stable after the initial bursts). However, their initial bursts happened after the elites. We ignore the changes of nonelite developers'

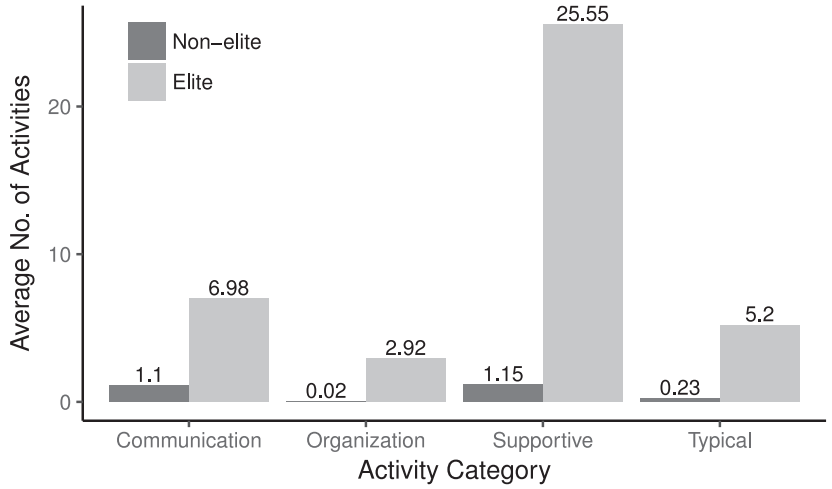


Fig. 5. Comparison of an individual elite and nonelite developer's average activity amount during a project-month.

organizational activities, since the nonelite do not have the privileges to perform nearly any of activities in this category.

To verify whether the effort distribution shifts of elite developers are common in our sampled projects, we test the differences in growth rates of activity categories as the next research question purposed.

4.2.2 Comparing Growth Rates of the Four Types of Activities. As mentioned in Section 3.6.2, we calculate the average monthly growth rates of activities per elite developer over the communicative, supportive, and typical activities[13] for each project. Thus, we have 20 growth rates for these three categories of activities. We then perform one-way ANOVA to test if there is any difference in growth rates.

The results show significant differences ($F_{(2,57)} = 8.452$, $p < 0.001$). We perform the post hoc analysis using the Tukey's HSD test to identify the differences between the three categories. The results indicate that the growth rates of typical activities are significantly lower than the growth rates of the other two (Typical vs. Communicative: $p = 0.002$, Typical vs. Supportive: $p = 0.002$). In fact, elite developers' typical activities even decrease over time (average growth rate = −1.63%). Though this number seems relatively small, it actually means an elite developer only does half of the technical work he or she used to do 3 years ago. Meanwhile, their communicative and supportive work doubled in the same period.

We do not perform the same ANOVA procedures to the nonelites' data for cross-group comparisons (i.e., elite vs. nonelite) due to practical constraints. In many months, the nonelites' activity counts are 0. Thus, calculating growth rates would lead to many "division by zero" problems. However, qualitatively, we could not observe any significant increases on the three types of nontechnical activities over time, while the numbers of nonelites' technical activities in each project-month do increase over time.

Answers to RQ_2. Based on the result of the one-way ANOVA test and Tukey's HSD test, we can answer RQ_2 as follows:

[13]For organizational activities, many months do not record such types of activities. This prohibits us from calculating the growth rate.

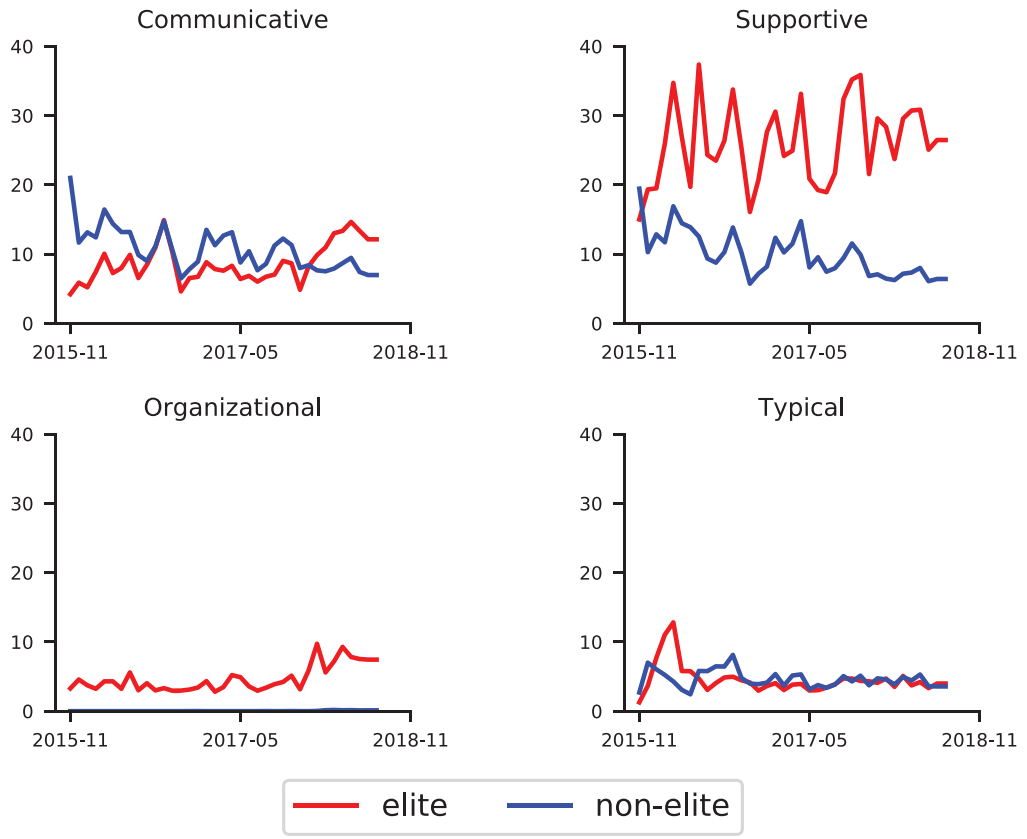


Fig. 6. Trends of individual developers' activities in the four activity categories of the *Tensorflow*.

With the progress of the project, an elite developer tends to put more effort into communicative and supportive activities, while significantly reducing his or her involvement in typical development activities.

4.3 RQ_3: Elite Developers' Activities' Impacts on Project Outcomes

We present our findings to $\mathbf{RQ_3}$ with a series regression models (Tables 5 and 6) characterizing relationships between the shares of activities in the three categories and the product productivity and quality indicators. These models are developed using the econometrics techniques discussed in Section 3.6.3. We also perform regression diagnostics to empirically examine the justification of using fixed effects models in model development. For all 12 regressions, fixed effects models are better choices than pooled OLS and random models.

Note that all regression models establish *correlations* only, rather than *causalities*. However, when interpreting the results, we may give some propositions implying possible but not definitive causalities, which is a common practice in data-driven research related to human and social factors [14, 21, 35]. These implied causalities should not be considered as established without further confirmatory studies [75].

4.3.1 Regression Results of Project Productivity. Table 5 summarizes the results of the regression models for the two project productivity indicators: the number of new commits of project i in

Table 5. Regression Models for Project Productivity

	Whole Sample		Subsample (Noncompany)		Subsample (Company)	
	New Commit	Bug Cycle Time	New Commit	Bug Cycle Time	New Commit	Bug Cycle Time
	Model P1 (β)	Model P2 (β)	Model P3 (β)	Model P4 (β)	Model P5 (β)	Model P6 (β)
	(SE)	(SE)	(SE)	(SE)	(SE)	(SE)
$\overline{S-Com_{im}}$	-155.96^{**}	-170.71	-125.07^{***}	-381.80	$-228.68*$	-16.44
	(49.20)	(143.96)	(28.11)	(212.09)	(92.85)	(202.66)
$\overline{S-Org_{im}}$	1.38	$-148.61*$	$-137.28*$	$-214.71*$	203.10	-153.85
	(121.25)	(54.65)	(70.24)	(129.6)	(223.47)	(187.49)
$\overline{S-Sup_{im}}$	-138.21^{***}	473.60^{***}	-91.25^{***}	553.17^{**}	-204.66^{**}	401.30^{**}
	(35.52)	(103.94)	(20.34)	(153.39)	(66.16)	(144.40)
Unobserved time-invariant effects (α_i)[¶]	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$
Multiple R^2	0.884	0.745	0.686	0.740	0.891	0.751
Adjusted Multiple R^2	0.880	0.736	0.673	0.730	0.887	0.742
$F^{\dagger}$	231.10^{***}	88.35^{***}	60.45^{***}	73.89	222.00^{***}	82.44^{***}

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

[¶]The unobserved time-invariant effects are a vector rather than a single coefficient. To keep the article concise, we do not include them; instead, we show the significant levels of them.

[†]For Models 1–2: degrees of freedom (DFs) are (23, 697); for Models 3–4: degrees of freedom are (12, 312); for Models 5–6: degrees of freedom are (14, 382).

Table 6. Regression Models for Project Quality

	Whole Sample		Subsample (Noncompany)		Subsample (Company)	
	New Bug Model Q1 (β) (SE)	Bug Fix Rate Model Q2 (β) (SE)	New Bug Model Q3 (β) (SE)	Bug Fix Rate Model Q4 (β) (SE)	New Bug Model Q5 (β) (SE)	Bug Fix Rate Model Q6 (β) (SE)
$\overline{S-Com_{im}}$	4.13	-2.24^{***}	1.43	-1.28^{***}	5.23	-3.11^{***}
	(5.26)	(0.35)	(3.91)	(0.34)	(9.59)	(0.61)
$\overline{S-Org_{im}}$	50.13^{***}	-0.63	6.48	-0.62	88.69^{***}	-0.35
	(12.97)	(0.87)	(9.77)	(0.85)	(23.09)	(1.47)
$\overline{S-Sup_{im}}$	18.31^{***}	1.12^{***}	-7.67^{**}	0.60^{*}	26.99^{***}	1.50^{***}
	(3.80)	(0.26)	(2.83)	(0.25)	(6.84)	(0.44)
Unobserved time-invariant effects (α_i)[¶]	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$	$-.-^{***}$
Multiple R^2	0.857	0.649	0.667	0.761	0.871	0.608
Adjusted Multiple R^2	0.853	0.637	0.654	0.752	0.866	0.594
$F^{\dagger}$	186.2^{***}	56.09^{***}	52.05^{***}	83.07^{***}	184.2^{***}	42.31^{***}

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

[¶]The unobserved time-invariant effects are a vector rather than a single coefficient. To keep the article concise, we do not include them; instead, we show the significant levels of them.

[†]For Models 1–2: degrees of freedom (DFs) are (23, 697); for Models 3–4: degrees of freedom are (12, 312); for Models 5–6: degrees of freedom are (14, 382).

month m ($NewC_{im}$) and the average bug cycle time of project i in month m (BCT_{im}). Models P1 and P2 use the data of all 20 sampled projects and thus represent whole sample regression results. Models P3 and P4 use the data of 9 non-company-sponsored projects, while models P5 and P6 use the data of 11 company-sponsored projects. Thus, models P3 through P6 represent subsample regression results. Below we describe what these models indicate.

A. Project Productivity—Whole Sample Regression Results

In Model P1, two independent variables ($\overline{S-Com_{im}}$, $\overline{S-Sup_{im}}$) are significant, and both have negative regression coefficients (−155.96, −138.21). This implies **negative correlations** between the effort elite developers put into communicative and supportive activities and the number of new commits in each project-month. A possible interpretation of the results is as follows. We already know that elite developers are still major contributors of source code. When they invest more effort into nontechnical activities, such as communicative and supportive ones, they may have less time to contribute to the source code; thus, the project may have fewer new commits (productivity loss).

In Model P2, for which the dependent variable is BCT_{im}, two independent variables ($\overline{S-Org_{im}}$, $\overline{S-Sup_{im}}$) are significant. $\overline{S-Org_{im}}$ has a negative regression coefficient (−148.61), while $\overline{S-Sup_{im}}$ has a positive coefficient (473.60). Obviously, there are **negative correlations** between elite developers' efforts in organizational activities and average bug cycle time in each project-month, and **positive correlations** between elite developers' efforts in supportive activities and average bug cycle time in each project-month. Given that the activities of managing bug fix and code review are in the "organizational" category (see Figure 2), more elite efforts in this category might help to shorten the bug cycle time. Meanwhile, similar to the results and interpretation for Model P1, performing more supportive activities may occupy elite developers' time on fixing bugs, and thus lead to longer bug cycle time (productivity loss). Since $\overline{S-Sup_{im}}$'s effect is much stronger than $\overline{S-Org_{im}}$'s and its shares are often much more than $\overline{S-Org_{im}}$'s (avg.: 0.61 vs. 0.03), we could expect an overall effect of longer bug cycle time (productivity loss).

B. Project Productivity—SubSample Regression Results.

The regression results in Models P3 through P6 are similar to those in Models P1 and P2 with some minor differences. Let us first have a look at the regression models based on non-company-sponsored projects' data (Models P3 and P4). In Model P3, $\overline{S-Org_{im}}$ becomes a significant variable, indicating that performing more organizational activities is also **negatively correlated** with the number of new commits in each project-month (productivity loss). In Model P4, the correlations between efforts on each category and the average bug cycle time in each project-month (BCT_{im}) are the same. For the regression models based on company-sponsored projects' data (Models P5 and P6), correlations in Model P5 are the same as those in Model P1. However, in Model P6, $\overline{S-Org_{im}}$ is no longer significant. A possible explanation may be that company-sponsored projects often have established routine bug fixing processes, and hence elite developers' mediation in this process is not as important.

In addition, the adjusted R^2s of Models P5 and P6 are higher than Models P3 and P4. Particularly, Model P5's is over 20% higher than Model P3's. These differences indicate that models built around the elite developers' activities explain larger proportions of variances of the data from company-sponsored projects than that of the data from non-company-sponsored projects.

4.3.2 Regression Results of Project Quality. We have briefly discussed the regression results of project productivity. Now, let us turn to the regression results of project quality. Table 6 summarizes the results of the regression models for the two project productivity indicators: the number

of new bugs of project i in month m ($NewC_{im}$) and the big fixed rate of project i in month m (BFR_{im}). Similarly, Models Q1 and Q2 use the data of all 20 sampled projects, thus representing whole-sample regression results. Models Q3 and Q4 use the data of 9 non-company-sponsored projects, while models Q5 and Q6 use the data of 11 company-sponsored projects. Thus, Models Q3 through Q6 represent subsample regression results.

A. Project Quality—Whole Sample Regression Results

In Model Q1, the quality indicator is $NewC_{im}$. There are two significant independent variables ($\overline{S-Org_{im}}$, $\overline{S-Sup_{im}}$); both have positive regression coefficients (50.13, 18.31). This indicates **positive correlations** between the effort elite developers dedicate to organizational and supportive activities and the number of new bugs found in each project-month. The interpretation of the results shall be similar to the above. We suspect that as the community grows, doing nontechnical work for supporting other community developers and users may make the elite have less time to work on code. Meanwhile, the project may receive a large amount of suggested changes and patches from nonelite developers, but their code may contain more bugs (quality loss).

In Model Q2, the quality indicator is BFR_{im}. Two independent variables are significant ($\overline{S-Com_{im}}$, $\overline{S-Sup_{im}}$). $\overline{S-Com_{im}}$'s coefficient is negative, signifying **negative correlations** between the effort elite developers put into communicative activities and each month's bug fix rate. Meanwhile, $\overline{S-Sup_{im}}$'s coefficient is positive, indicating **positive correlations** between the elites' efforts in supportive activities and each month's bug fix rate. Interpreting such correlations may be a bit tricky. For the negative correlations between $\overline{S-Com_{im}}$ and BFR_{im}, we can interpret them in a way similar to the previous ones. The positive correlations between $\overline{S-Sup_{im}}$ and BFR_{im} may suggest that by putting more effort into supportive activities, elite developers help to make the defect removal process more effective.[14]

Since $\overline{S-Com_{im}}$'s share is only about one-fourth of $\overline{S-Sup_{im}}$'s (avg.: 0.16 vs. 0.61) and its negative coefficient is just twice of $\overline{S-Sup_{im}}$'s (-2.24 vs. 1.12), we could expect an overall on average effect of a higher bug fix rate (quality gain).

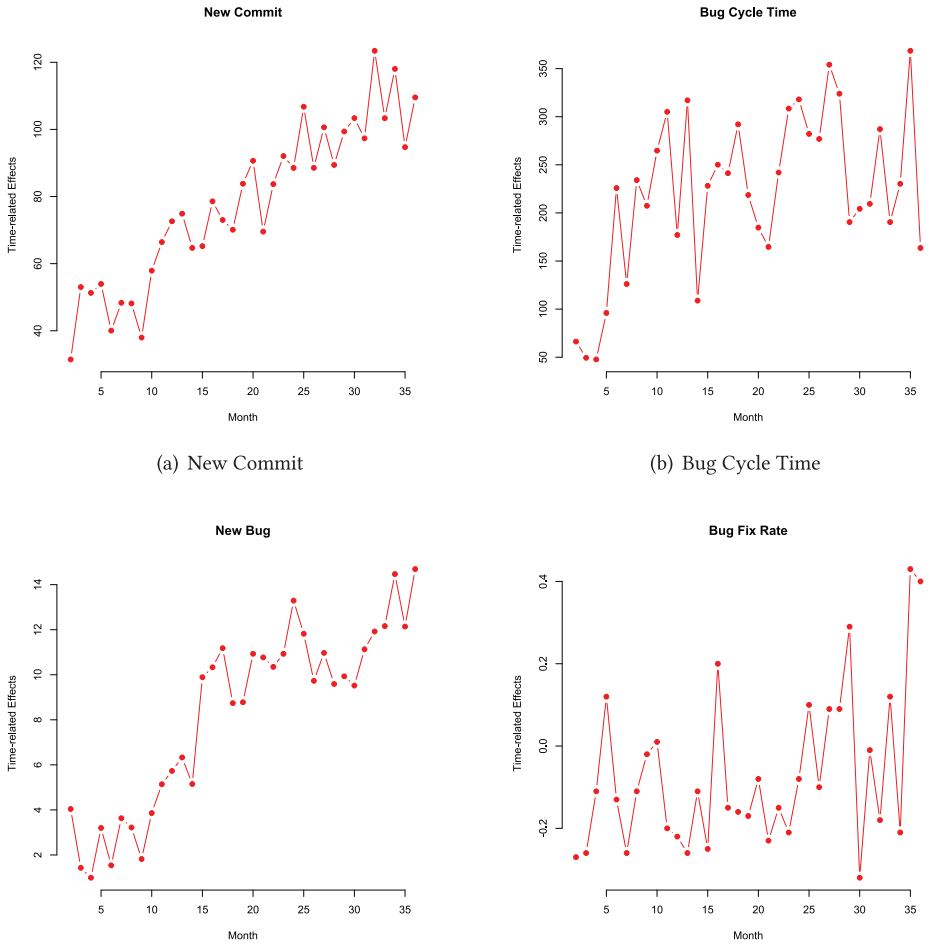
B. Project Quality—SubSample Regression Results

Again, we split the whole-sample dataset into two subsample datasets according to whether a project is sponsored by a commercial company and then develop regression models using them. Models Q3 and Q4 are based on non-company-sponsored projects' data. In Model Q3, only $\overline{S-Sup_{im}}$ is significant. Efforts on organizational activities are not negatively correlated with the number of new bugs in each project-month. Model Q4 is similar to Model Q2. In general, the effects in Models Q5 and Q6 are quite similar to those in Models Q1 and Q2.

4.3.3 Time-Related Effects. To further explore the time-related effects, we perform time-fixed effects testing for the four whole-sample models (Models P1, P2 in Table 5 and Models Q1, Q2 in Table 6).

For Model P1, where the number of new commits in each project-month is the dependent variable, the time-fixed effects model is significant ($F(38, 662) = 1.59$, $p = 0.02$). However, the effects are less significant (adjusted $R^2 = 0.01$). Further examination of the time-fixed effects shows that the time-related effects are positive and exhibit an increasing trend (Figure 7(a)). This indicates that the number of new commits is less associated with elite developer activities in the later phases of the project. However, for Model P2, where the bug cycle time in each project-month is the

[14]Recall that BFR_{im} is indeed a quality process metrics (Section 3.4).



(a) New Commit

(b) Bug Cycle Time

(c) New Bug

(d) Bug Fix Rate

Fig. 7. Changes of time-related effects in the regression models.

dependent variable, the time-fixed effects model is not significant ($F(38, 662) = 1.80$, $p < 0.01$). The effects are very small (adjusted $R^2 = 0.01$). The time-related effects have slightly patterns (Figure 7(b)).

For Model Q1, where the number of new bugs in each project-month is the dependent variable, the time-fixed effects model is significant ($F(38, 662) = 3.29$, $p < 0.001$). The results are similar to the first one (Figure 7(a)). The time-related effects are positive and increasing in general (Figure 7(c)), indicating that the impact of elite developers' activities on the number of new bugs reported is shrinking over time. For Model P2, where the bug fix rate in each project-month is the dependent variable, the time-fixed effects model is not significant ($F(38, 662) = 1.07$, $p = 0.36$). No meaningful effect could be detected (adjusted $R^2 = 0.00$). The time-related effects may be irrelevant to this quality indicator (Figure 7(d)).

The above analyses reveal that although the time-related effects are significant, the project-specific fixed effects models are much stronger than the time-related effects for all dependent variables.

Answers to RQ_3. Based on the above results, we can answer RQ_3 as follows:

Elite developers' effort distributions have significant correlations with project outcomes.

(1) *Project Productivity: (a) Efforts on communicative and supportive activities are negatively correlated with the project productivity in terms of the number of new commits in each project-month (productivity loss). (b) Efforts on organizational activities are positively correlated with project productivity in terms of the average bug cycle time in each project-month; however, efforts on supportive activities have much stronger negative effects. The overall effects are negative (productivity loss).*
(2) *Project Quality: (a) Efforts on organizational and supportive activities are positively correlated with the number of newly found bugs in each project-month (quality loss). (b) Efforts on communicative activities are negatively correlated with the bug fix rate in each project-month; however, efforts on supportive activities have positive effects. Combine them together, the overall effects are likely to be positive (quality gain).*
(3) *Except for the bug fix rate, time effect analyses show that the impacts exhibit some decreasing trends with the progress of the project, which may result from the increasing proportion of nonelite developers' contributions in the latter stages of the project.*
(4) *In general, compared with the company-sponsored projects, effort distribution correlations with project outcomes are less significant for non-company-sponsored projects.*

5 DISCUSSION

5.1 Discussion of the Findings

First of all, our results and findings confirm the important roles of elite developers in open source development. As the result of RQ_1 shows, elite developers engaged in the majority of the projects' activities, though they only account for a small proportion of contributors in the larger community. Except for communicative activities, elite developers account for over 50% activities in all the other three categories. The results confirm prior literature dating back to the early 2000s [26, 62, 102]. We can conclude that open source projects are still largely driven by a small number of elite members after over 20 years of evolution. While such high concentrations may ensure bottom-line project outcomes, such situations may not be optimal for the long-term health of a project [25]. Engaging the nonelite users' participation through mechanism and technology innovation is still a challenge [82]. To sum up, the results of RQ_1 confirm and extend the findings in the prior study focused on developers' workload such as [102]. From the role-specific perspective (elite vs. nonelite), our work reveals that the imbalance of workload not only exists in technical contributions but also happens in nontechnical activities, where situations are even worse.

Second, the results and findings of RQ_2 show that in most of the sampled projects, elite developers performed different activities over time. The activity shifting indicates the elite developers' role transitions with the growth of the project and the community. Organizational behavior theorists often argue that such transitions may be risky and troublesome for both individuals and organizations [4, 64]. Consider a developer who, while initiating the project, was heavily involved in contributions on source code artifacts, and through these activities intends to build his or her technical competencies and establish his or her community reputation. However, as the project and community grew, he or she found herself having to spend more time supporting and communicating with novice contributors and users. Imposing additional burdens of management is usually less effective without necessary training and self-motivation. Developers may vary in their personal perceptions about how they would develop their competencies [52], but, at least in the software

engineering community, unfortunately this issue has not received any attention. Future research is necessary to address the issues related to such role transitions.

The results and findings of **RQ$_3$** reveal relationships between elite developers' effort distributions and project outcomes. In general, there are some negative associations. For three out of four project outcome indicators ($NewC_{im}$, BCT_{im}, and $NewB_{im}$), our results suggest that increased efforts into communicative, organizational, and supportive work are negatively correlated with the project outcomes. We suspect the negative association may due to the increased involvement of nonelite developers and the work shift of elite developers. Typically, as a project's community grows, external nonelite developers may contribute to the project in various ways, including suggesting features, reporting bugs, or submitting their own patches [26, 46]. Since those nonelite developers often do not have a comparable level of technical expertise, their code could be more buggy, and thus may lead to lower software quality [1]. Their contribution requires vast attention and support from the elite group, which inevitably reduces direct elite group effort on typical development. However, for the last project outcome indicator (BFR_{im}), our results show that the elites' efforts in supportive work have positive correlations with project quality. A possible explanation is that the efforts in supportive activities help to maintain a good defect removal process, and thus improve the bug fix rate in each project-month.

RQ$_3$'s findings, if put together, describe a dilemma that elite developers often have to face in their projects. With the growth of their projects, they need to spend more time on nontechnical tasks, which forces them to reduce their efforts on technical contributions. Since their technical activities still account for a majority of the project's typical development work (see Table 4), the project would also experience some productivity and quality loss. Yet, nontechnical work has its own value: it perhaps helps to maintain a project's work processes (e.g., defect removal process) and pays off with some quality gains.

Another finding worth noting is the difference between non-company-sponsored projects and company-sponsored projects. Particularly, our **RQ$_3$**'s results indicate that company-sponsored projects tend to be more influenced by their elite developers' effort distributions. This is not surprising, since such projects often rely on a small amount of full-time employees as the elite developers, and it also empirically confirms early qualitative results [92]. Our findings support that with the involvement of industrial companies, economic incentives/supports and potential job opportunities lead to more displays of contributors' technical skills in company-sponsored projects [55, 61]. Thus, elite developers more actively influence these projects. Moreover, GitHub recently provided a "sponsors" feature to financially support OSS developers; we may expect increased influence on conventional projects, which are largely based on voluntary contribution [55].

To sum up, our work not only confirms the empirical observations of developers' activities in open source communities but also provides new findings and insights that shed light on future research. For example, we observe the elite developers' role transitions from the shifting of their work concentrations, whereas supporting such transitions has not yet been investigated. Additionally, we identify the impacts of effort distributions over the four broader categories on project outcomes. As far as our best current knowledge, this is the first piece of empirical evidence on this topic. How to leverage the findings to bring better project outcomes requires follow-up research.

5.2 Practical Implications

Our findings suggest immediate practical implications. Similar to the results in [102], we found that the imbalance of workload is prevalent and leads to heavy burdens on a few elite developers. In addition to the unbalanced technical workload identified in that prior study, we further found that situations are even worse for nontechnical workload (communicative, organizational, and supportive). The imbalances of nontechnical workload are often at a huge order of magnitude

(see Section 4.1). By examining the projects in our sample, we could say that these imbalances at least partially result from the fact that the increase of elite developers often fails to keep pace with project growth, which cannot be spontaneously fixed in the evolution of projects (see Section 4.2). This is especially true given that many open source projects are conservative when it comes to guaranteeing a member the permissions to perform some administrative tasks. While the open source ideology is fairly progressive, its management structures are perhaps somewhat preindustrial; i.e., only a few elites share most of the authority and power in the community [20, 78, 91]. Decentralizing such authority and power, particularly related to routine work, might be an option. It would not only alleviate elite developers' burdens but also provide extra incentives for ordinary members in communities to contribute [73]. Moreover, because the turnover of the core reviewers is high and rapid [86], allowing some nonelite developers to share some elite developers' routine duties would help offset the negative impacts of their turnover. In fact, decentralizing and delegating some proportions of elite developers' work has been recognized by practitioners. Recently, GitHub[15] acknowledged that inadequate governance and excessive workload of a few core individuals are two major threats to project sustainability. They also mentioned that allowing and guiding ordinary members to run the project is critical to address these threats. Therefore, doing so would be fairly easy in the near future, particularly in the case that GitHub has already committed to provide facilities to help projects to implement such decentralization and delegation.

Second, $\mathbf{RQ_3}$'s findings also highlight that relationships between project outcomes and elite developers' efforts in nontechnical tasks are more significant for company-sponsored projects. As opposed to non-company-sponsored projects, company-sponsored projects more or less inherit the management practices of the corporate world. Elite developers tend to be trapped more in routine work. In his dissertation [92], Wagstrom has shown that the vertical integration between companies and open source communities would inevitably lead to increases in unnecessary communicative and organizational practices. Given the limited time and attention resources of developers, these unnecessary nontechnical practices may hurt a project's productivity. Thus, he recommended focusing on communication "meeting individual coordination requirements." According to our results, his recommendation is still valid. From a company's perspective, avoiding "copying" their internal governing structures may be necessary, even for the projects they dominate [37, 77, 97].

5.3 Design Implications

With the growth of the project, elite developers often have to put more effort into communicative and supportive tasks. Our study reveals that such a shifting of work may have negative impacts on project outcomes. As we discussed in Section 5.1, since these tasks are often necessary and cannot be ignored, building software tools to assist or partially free elite developers may be a good solution.

Building such tools is feasible, especially since readily available technologies exist for many organizational and supportive activities. For instance, *Assigned* and *Unassigned* are two main events in the organizational activity category (see Figure 2). The main time cost for them is to identify the external assignee. These tasks can be easily automated with tools [3]. The supportive work can be divided into two sets—maintenance and documentation. For many raw activities associated with maintenance, there are ready-to-use automated tools built by researchers. For example, the *CreateTag* can be automated using techniques such as [19]. Automatic subscribe and unsubscribe can be realized through learning users' characteristics [12]. For documentation tasks, there are

[15]Let's talk about open source sustainability: https://github.blog/2019-01-17-lets-talk-about-open-source-sustainability/.

many metric-based or machine learning techniques ready for use [60, 101], thus automating some *MarkedAsDuplicated* and *UnMarkedAsDuplicated* tasks.

Current technologies may be less mature for helping elite developers on communicative tasks. As shown in Figure 2, the communicative category contains four raw GitHub activities: *Reference*, *Edit*, *IssueComment*, and *CommitComment*. For some activities related to *Reference*, researchers have developed techniques for automating them. For example, when mentioning somebody to fix an issue, the bug-fixer recommendation technique developed by Kim et al. [49] may be directly applied to identify the target of the mentioning. *CommentDeleted* tasks also can be automated. For example, a disruptive message by a member can be automatically deleted by a GitHub bot app equipped with advanced sentiment analysis techniques. Building automated tools for *IssueComment* and *CommitComment* requires some advanced techniques on abstractive semantic summarization and text generation, which are far from mature even in the Natural Language Processing community [57, 59, 96].

While there are many available techniques, most (if not all) of them have never been used by practitioners. This may be because such techniques have not been integrated into elite developers' normal workflows. As Terry Winograd and his colleagues [98] pointed out in their influential book *Understanding Computers and Cognition: A New Foundation for Design*, a computing application must be integrated into users' workflow in a nonintrusive way to gain widespread acceptance. Moreover, our results also call for innovative workflow designs that focus on not only technical contributions (e.g., multiple-committer model in [84]) but also those optimized for the full spectrum of activities.

5.4 Recommendations

Our findings and the above discussion can be summarized into recommendations for practitioners and researchers.

Recommendations for open source practitioners are:

- *Open source projects may consider decentralizing the administrative authorities and powers related to routine tasks.*
- *Project members should focus on communication "meeting individual coordination requirements."*
- *Projects sponsored by companies should avoid copying their sponsors' internal governing structures.*

We can also consider future research (including tool design and implementation) efforts with the following possible challenges:

- *Further understanding of developer activities that takes an integrated perspective combining both role-based relationships and different types of activities beyond technical contributions*
- *Mechanism and workflow designs for broadening participation in and sharing nontechnical responsibilities*
- *Tool support for relieving elite developers from routine administrative burdens by synthesizing existing techniques to their routine workflow*

5.5 Threats to Validity

As with any empirical studies, our study is not free of threats to validity. We briefly discuss them from three perspectives.

First, from the perspective of **construct validity**, we are confident that there is no significant threat. Our study involves six primary constructs, which are four categories of GitHub activities,

and project productivity and quality (each with two metrics, total four metrics). All of their definitions and operationalizations are based on prior literature. For the four activity categories, we follow the standard procedure to develop the mappings between raw GitHub activities and these categories. The two project outcomes are adapted from the literature, and each is measured by two distinct indicators. By using multiple indicators for one project outcome construct, our study avoids oversimplifying the concept of "productivity" and "quality" and brings new insights. Thus, we have the confidence that most of the threats to construct validity have been avoided.

Second, from the perspective of **internal validity**, we took multiple measures to ensure that the data collection process avoids most of the perils summarized in [11, 47]. For example, all subjected projects are all large ones with established governing structure and practices and use pull-requests to manage members' contributions. The data used in the study are objective human activity records collected from online repositories. The analysis processes are unbiased. We use mature, widely used analysis techniques and empirically justify the use of the fixed effects models in panel regressions.

One potential threat is that our data comes from one source: GitHub. But doing so has its methodological justifications. While we acknowledge that the development trace data could be in multiple other channels such as email, IRC, forums, and so on, an unfortunate fact is that not all of them are publicly available. In fact, for the 20 projects studied in this article, none of them has all the channels of data available. If we use multiple data sources for some projects but a single data source for the rest, guaranteeing the fair comparisons among them could be impossible. Moreover, selectively using multiple data sources would pose serious threats to the "construct validity" because establishing the mapping between activities and categories would require different protocols when crossing data sources. Thus, having weighed the gain and loss of using multiple data sources, we decide only to use GitHub data; this at least guarantees consistency at the methodological level, which is a basic requirement for any scientific inquiry [13, 31, 53]. Thus, we view not using multiple data sources as a limitation, but not a serious threat to internal validity, as pointed out by Margaret-Anne Storey in her *ICSE'19* keynote [83].

Third, from the perspective of **external validity**, we admit that while our results may not be generalizable to all open source projects, the sampled projects represent a wide range of projects regarding the application domains. They also form a balanced sample of non-company-sponsored and company-sponsored projects. One potential limitation is that all 20 projects are large ones. We urge caution, however, for applying our findings in the context of small or medium-sized open source projects.

6 CONCLUSION

From the unique perspective of the division of labor in open source projects, developers can be classified into elite and nonelite according to their managerial privileges. While elite developers' important role in open source development has been long known in the software engineering literature, their activities have not yet been thoroughly investigated. Using fine-grained event data of 20 open source projects, our study paints a detailed and dynamic picture of elite developers' activities in four sense-making, high-level categories, as well as their activities' impact on project outcomes in terms of project productivity and product quality.

Our study yields a set of findings. First, our study confirms the essential roles of elite developers. Their activities account for the majority across all four types of broader activity categories: communicative, organizational, supportive, and typical. Second, our study reveals that elite developers' activities shift to the "project management" tasks from "technical" work. We observe that communicative and supportive activities increase much faster than typical development activities. Third, elite developers' effort distributions have significant correlations with project productivity

and quality outcomes. When they put more effort into communicative and supportive work, a project's productivity (measured by the number of new commits ($NewC_{im}$) and bug cycle time (BCT_{im}) in each project-month) is likely to decrease. Additionally, a project's quality (measured by the number of new bugs in each project-month ($NewB_{im}$)) is negatively associated with their activities on organizational and supportive tasks. Another indicator—bug fix rate in each project-month (BFR_{im})—is positively correlated with supportive activities, and thus may increase when the elite developers put more efforts toward support. These findings reveal a complicated picture of elite developers' effort distributions and partially indicate a dilemma faced by many OSS elite developers—i.e., with the growth of a project, its elite developers have to conduct more communicative and supportive work. We discuss the practical and design implication of the study.

For future work, we plan to continue the focus on elite developers. We plan to replicate this study with a larger sample of projects and go one step further to explore the contextualized, individual differences among elite developers. Currently, the analysis unit is at the project level; we also plan to extend the study by performing analyses at multiple levels, e.g., at the individual level or the ecosystem level. Moreover, since there are project outcomes beyond productivity and quality, we plan to explore some alternative outcomes, particularly those related to social and human development (e.g., the growth of newcomers). We will also design and implement tools to (at least partially) free elite developers from increasing communicative and supportive tasks, allowing them to maximize the impacts of their technical leadership in projects. Besides, we would like to pursue novel mechanism and workflow design balancing both technical and nontechnical contributions among all project members.

ACKNOWLEDGMENTS

We would like to thank the associate editor and anonymous reviewers for their insightful review comments and suggestions.

REFERENCES

[1] Mark Aberdour. 2007. Achieving quality in open-source software. *IEEE Software* 24, 1 (2007), 58–64.

[2] Juan Jose Amor, Gregorio Robles, and Jesus M. Gonzalez-Barahona. 2006. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research (EDSER'06)*. ACM, New York, NY, 3–6. DOI: https://doi.org/10.1145/1139113.1139116

[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. ACM, New York, NY, 361–370. DOI: https://doi.org/10.1145/1134285.1134336

[4] Blake Ashforth. 2000. *Role Transitions in Organizational Life: An Identity-based Perspective*. Routledge.

[5] Sogol Balali, Igor Steinmacher, Umayal Annamalai, Anita Sarma, and Marco Aurelio Gerosa. 2018. Newcomers' barriers. . . Is that all? An analysis of mentors' and newcomers' barriers in OSS projects. *Computer Supported Cooperative Work (CSCW)* 27, 3 (Dec 2018), 679–714. DOI: https://doi.org/10.1007/s10606-018-9310-8

[6] Sebastian Baltes and Stephan Diehl. 2018. Towards a theory of software development expertise. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'18)*. Association for Computing Machinery, New York, NY, 187–200. DOI: https://doi.org/10.1145/3236024.3236061

[7] Gary S. Becker and Kevin M. Murphy. 1992. The division of labor, coordination costs, and knowledge. *Quarterly Journal of Economics* 107, 4 (1992), 1137–1160.

[8] Christian Bird. 2011. Sociotechnical coordination and collaboration in open source software. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance (ICSM'11)*. IEEE, 568–573. DOI: https://doi.org/10.1109/ICSM.2011.6080832

[9] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code!: Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ACM, 4–14.

[10] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. 2008. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'08/FSE-16)*. ACM, New York, NY, 24–35. DOI: https://doi.org/10.1145/1453101.1453107

[11] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu. 2009. The promises and perils of mining git. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR'09)*. 1–10. DOI: https://doi.org/10.1109/MSR.2009.5069475

[12] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *Proceedings of the 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE'13)*. 188–197. DOI: https://doi.org/10.1109/ISSRE.2013.6698918

[13] Kenneth S. Bordens and Bruce B. Abbott. 2002. *Research Design and Methods: A Process Approach*. McGraw-Hill.

[14] danah boyd and Kate Crawford. 2012. Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon. *Information, Communication & Society* 15 (2012), 662–679.

[15] Robert L. Brennan and Dale J. Prediger. 1981. Coefficient kappa: Some uses, misuses, and alternatives. *Educational and Psychological Measurement* 41, 3 (1981), 687–699.

[16] Felix C. Brodbeck. 1994. Software-Entwicklung: Ein Tätigkeitsspektrum mit vielfältigen Kommunikations-und Lernanforderungen. In *Produktivität und Qualität in Software-Projekten: Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung*, Felix C. Brodbeck and Michael Frese (Eds.). Oldenbourg-Verlag, 13–34.

[17] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2012. Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE'12)*. ACM, New York, NY, Article 44, 11 pages. DOI: https://doi.org/10.1145/2393596.2393647

[18] Marcelo Cataldo and James D. Herbsleb. 2012. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering* 39, 3 (2012), 343–360.

[19] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*. ACM, New York, NY, 396–407. DOI: https://doi.org/10.1145/3106237.3106285

[20] Benjamin Collier, Moira Burke, Niki Kittur, and Robert E. Kraut. 2010. Promoting good management: Governance, promotion, and leadership in open collaboration communities. In *Proceedings of the Thirty First International Conference on Information Systems (ICIS'10)*. 220.

[21] Josh Cowls and Ralph Schroeder. 2015. Causation, correlation, and big data in social science research. *Policy & Internet* 7 (2015), 447–472. DOI: https://doi.org/10.1002/poi3.100

[22] Yves Croissant and Giovanni Millo. 2008. Panel data econometrics in R: The PLM package. *Journal of Statistical Software* 27, 2 (2008), 1–43. DOI: https://doi.org/10.18637/jss.v027.i02

[23] Kevin Crowston, Hala Annabi, James Howison, and Chengetai Masango. 2004. Effective work practices for software engineering: Free/libre open source software development. In *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research (WISER'04)*. ACM, New York, NY, 18–26. DOI: https://doi.org/10.1145/1029997.1030003

[24] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. *First Monday* 10, 2 (2005). DOI: https://doi.org/10.5210/fm.v10i2.1207

[25] Kevin Crowston and James Howison. 2006. Assessing the health of open source communities. *Computer* 39, 5 (2006), 89–91.

[26] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2008. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys* 44, 2, Article 7 (March 2008), 35 pages. DOI: https://doi.org/10.1145/2089125.2089127

[27] Kevin Crowston, Kangning Wei, Qing Li, and James 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)*. IEEE, Article 118, 118:1–118:10 pages.

[28] Daniel Alencar da Costa, Uirá Kulesza, Eduardo Aranha, and Roberta Coelho. 2014. Unveiling developers contributions behind code commits: An exploratory study. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14)*. ACM, New York, NY, 1152–1157. DOI: https://doi.org/10.1145/2554850.2555030

[29] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW'12)*. Association for Computing Machinery, New York, NY, 1277–1286. DOI: https://doi.org/10.1145/2145204.2145396

[30] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline P. de Vries. 2010. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE'10)*. ACM, New York, NY, 275–284. DOI: https://doi.org/10.1145/1806799.1806842

[31] Martyn Denscombe. 2014. *The Good Research Guide: For Small-Scale Social Research Projects*. McGraw-Hill.

[32] Luis Felipe Dias, Igor Steinmacher, and Gustavo Pinto. 2018. Who drives company-owned OSS projects: Internal or external members? *Journal of the Brazilian Computer Society* 24, 1 (2018), 16.
[33] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling static analyses at Facebook. *Communications of the ACM* 62, 8 (July 2019), 62–70. DOI: https://doi.org/10.1145/3338112
[34] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368.
[35] Liran Einav and Jonathan Levin. 2014. Economics in the age of big data. *Science* 346 (2014), 1243089. DOI: https://doi.org/10.1126/science.1243089
[36] Kristin E. Flegal and Michael C. Anderson. 2008. Overthinking skilled motor performance: Or why those who teach can't do. *Psychonomic Bulletin & Review* 15, 5 (2008), 927–932.
[37] Matt Germonprez, Julie E. Kendall, Kenneth E. Kendall, Lars Mathiassen, Brett Young, and Brian Warner. 2016. A theory of responsive design: A field study of corporate engagement with open source communities. *Information Systems Research* 28, 1 (2016), 64–83.
[38] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. 2015. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE' 15)*. IEEE Press, 358–368. DOI: https://doi.org/10.1109/ICSE.2015.55
[39] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2011. "Not My Bug!" and other reasons for software bug report reassignments. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work (CSCW'11)*. Association for Computing Machinery, New York, NY, 395–404. DOI: https://doi.org/10.1145/1958824.1958887
[40] Marvin Hanisch, Carolin Haeussler, Stefan Berreiter, and Sven Apel. 2018. Developers' progression from periphery to core in the Linux kernel development project. In *Academy of Management Proceedings*, Vol. 2018. Academy of Management, Briarcliff Manor, NY 10510, 14263.
[41] James Howison and Kevin Crowston. 2014. Collaboration through open superposition: A theory of the open source way. *Management Information Systems Quarterly* 38, 1 (2014), 29–50.
[42] Federico Iannacci. 2005. Coordination processes in open source software development: The Linux case study. *Emergence: Complexity & Organization* 7, 2 (2005), 21–31. DOI: https://doi.org/10.emerg/10.17357.390b9a40a2b742dd4f68754179cb8714
[43] Chris Jensen and Walt Scacchi. 2007. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*. IEEE Computer Society, Washington, DC, 364–374. DOI: https://doi.org/10.1109/ICSE.2007.74
[44] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 2011. The onion patch: Migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE'11)*. Association for Computing Machinery, New York, NY, 70–80. DOI: https://doi.org/10.1145/2025113.2025127
[45] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*. IEEE Press, Piscataway, NJ, 164–174. DOI: https://doi.org/10.1109/ICSE.2017.23
[46] Nicolas Jullien, Klaas-Jan Stol, and James Herbsleb. 2019. A preliminary theory for open source ecosystem microeconomics. *arXiv preprint arXiv:1905.05985* (2019).
[47] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*. Association for Computing Machinery, New York, NY, 92–101. DOI: https://doi.org/10.1145/2597073.2597074
[48] Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. 2012. Do faster releases improve software quality? An empirical case study of Mozilla Firefox. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR'12)*. IEEE Press, 179–188.
[49] Dongsun Kim, Yida Tao, Sunghun Kim, and Andreas Zeller. 2013. Where should we fix this bug? A two-phase recommendation model. *IEEE Transactions on Software Engineering* 39, 11 (2013), 1597–1610.
[50] Sunghun Kim and E. James Whitehead. 2006. How long did it take to fix bugs? In *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR'06)*. Association for Computing Machinery, New York, NY, 173–174. DOI: https://doi.org/10.1145/1137983.1138027
[51] Stefan Koch and Georg Schneider. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal* 12, 1 (2002), 27–42.
[52] Ellen Ernst Kossek, Karen Roberts, Sandra Fisher, and Beverly Demarr. 1998. Career self-management: A quasi-experimental assessment of the effects of a training intervention. *Personnel Psychology* 51, 4 (1998), 935–960.
[53] Chakravanti Rajagopalachari Kothari. 2004. *Research Methodology: Methods and Techniques*. New Age International.

[54] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. ACM, New York, NY, 492–501. DOI: https://doi.org/10.1145/1134285.1134355

[55] Josh Lerner and Jean Tirole. 2002. Some simple economics of open source. *Journal of Industrial Economics* 50, 2 (2002), 197–234.

[56] Ytzhak Levendel. 1990. Reliability analysis of large software systems: Defect data modeling. *IEEE Transactions on Software Engineering* 16, 2 (1990), 141–152.

[57] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'16)*. 110–119.

[58] Bin Lin, Gregorio Robles, and Alexander Serebrenik. 2017. Developer turnover in global, industrial open source projects: Insights from applying survival analysis. In *Proceedings of the 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE'17)*. IEEE, 66–75.

[59] Fei Liu, Jeffery Flanigan, Sam Thomson, Norman Smith, and Noah A. Sadeh. 2015. Toward abstractive summarization using semantic representations. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'15)*. 1077–1086.

[60] D. V. Luciv, D. V. Koznov, George A. Chernishev, Andrey N. Terekhov, K. Yu Romanovsky, and D. A. Grigoriev. 2018. Detecting near duplicates in software documentation. *Programming and Computer Software* 44, 5 (2018), 335–343.

[61] Jennifer Marlow and Laura Dabbish. 2013. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW'13)*. Association for Computing Machinery, New York, NY, 145–156. DOI: https://doi.org/10.1145/2441776.2441794

[62] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering Methodology* 11, 3 (July 2002), 309–346. DOI: https://doi.org/10.1145/567793.567795

[63] Audris Mockus and James D. Herbsleb. 2002. Expertise browser: A quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)*. Association for Computing Machinery, New York, NY, 503–512. DOI: https://doi.org/10.1145/581339.581401

[64] Nigel Nicholson. 1984. A theory of work role transitions. *Administrative Science Quarterly* 29, 2 (1984), 172–191.

[65] Siobhan O'Mahony and Fabrizio Ferraro. 2007. The emergence of governance in an open source community. *Academy of Management Journal* 50, 5 (2007), 1079–1106.

[66] Brian T. Pentland and Martha S. Feldman. 2005. Organizational routines as a unit of analysis. *Industrial and Corporate Change* 14, 5 (2005), 793–815.

[67] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. 2019. Going farther together: The impact of social capital on sustained participation in open source. In *Proceedings of the 41st International Conference on Software Engineering (ICSE'19)*. IEEE Press, 688–699. DOI: https://doi.org/10.1109/ICSE.2019.00078

[68] R Development Core Team. 2008. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from http://www.R-project.org ISBN 3-900051-07-0.

[69] Foyzur Rahman and Premkumar Devanbu. 2011. Ownership, experience and defects: A fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. Association for Computing Machinery, New York, NY, 491–500. DOI: https://doi.org/10.1145/1985793.1985860

[70] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in Github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. Association for Computing Machinery, New York, NY, 155–165. DOI: https://doi.org/10.1145/2635868.2635922

[71] Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12, 3 (1999), 23–49.

[72] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. 2008. Open source software peer review practices: A case study of the apache server. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*. ACM, New York, NY, 541–550. DOI: https://doi.org/10.1145/1368088.1368162

[73] Jeffrey A. Roberts, Il-Horn Hann, and Sandra A. Slaughter. 2006. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects. *Management Science* 52, 7 (2006), 984–999.

[74] Bertil Rolandsson, Magnus Bergquist, and Jan Ljungberg. 2011. Open source in the firm: Opening up professional practices of software development. *Research Policy* 40, 4 (2011), 576–587.

[75] Mike Savage and Roger Burrows. 2007. The coming crisis of empirical sociology. *Sociology* 41, 5 (2007), 885–899. DOI: https://doi.org/10.1177/0038038507080443

[76] Walt Scacchi. 2007. Free/open source software development. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE'07)*. ACM, New York, NY, 459–468. DOI: https://doi.org/10.1145/1287624.1287689

[77] Mario Schaarschmidt, Gianfranco Walsh, and Harald F. O. von Kortzfleisch. 2015. How do firms influence open source software communities? A framework and empirical analysis of different governance modes. *Information and Organization* 25, 2 (2015), 99–114.

[78] Sonali K. Shah. 2006. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science* 52, 7 (2006), 1000–1014.

[79] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken-ichi Matsumoto. 2013. Studying re-opened bugs in open source software. *Empirical Software Engineering* 18, 5 (2013), 1005–1042.

[80] Sabine Sonnentag. 1995. Excellent software professionals: Experience, work activities, and perception by peers. *Behaviour & Information Technology* 14, 5 (1995), 289–299.

[81] Sabine Sonnentag. 1998. Expertise in professional software design: A process study *Journal of Applied Psychology* 83, 5 (1998), 703.

[82] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW'15)*. Association for Computing Machinery, New York, NY, 1379–1392. DOI: https://doi.org/10.1145/2675133.2675215

[83] Margaret-Anne Storey. 2019. Publish or perish: Questioning the impact of our research on the software developer. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE'19)*. IEEE Press, Piscataway, NJ, 2–2. DOI: https://doi.org/10.1109/ICSE-Companion.2019.00021

[84] Xin Tan. 2019. Reducing the workload of the Linux kernel maintainers: Multiple-committer model. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*. Association for Computing Machinery, New York, NY, 1205–1207. DOI: https://doi.org/10.1145/3338906.3342490

[85] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. 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*. ACM, 644–655.

[86] Perry van Wesel, Bin Lin, Gregorio Robles, and Alexander Serebrenik. 2017. Reviewing career paths of the openstack developers. In *Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME'17)*. IEEE, 544–548.

[87] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. 2016. The sky is not the limit: Multitasking across GitHub projects. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. Association for Computing Machinery, New York, NY, 994–1005. DOI: https://doi.org/10.1145/2884781.2884875

[88] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. Stackoverflow and GitHub: Associations between software development and crowdsourced knowledge. In *Proceedings of the 2013 International Conference on Social Computing (SocialCom'13)*. IEEE, 188–195.

[89] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark G. J. van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. 2015. Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'15)*. Association for Computing Machinery, New York, NY, 3789–3798. DOI: https://doi.org/10.1145/2702123.2702549

[90] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*. Association for Computing Machinery, New York, NY, 805–816. DOI: https://doi.org/10.1145/2786805.2786850

[91] Georg Von Krogh and Eric Von Hippel. 2006. The promise of research on open source software. *Management Science* 52, 7 (2006), 975–983.

[92] Patrick Wagstrom. 2009. *Vertical Interaction in Open Software Engineering Communities*. PhD dissertation. Carnegie Mellon University.

[93] Patrick Wagstrom, Corey Jergensen, and Anita Sarma. 2012. *Roles in a Networked Software Development Ecosystem: A Case Study in GitHub*. Technical Report. TR-UNL-CSE-2012-0006. Department of Computer Science & Engineering, University of Nebraska-Lincoln.

[94] Ronald L. Wasserstein and Nicole A. Lazar. 2016. The ASA's statement on p-values: Context, process, and purpose. *American Statistician* 70, 2 (2016), 129–133.

[95] E. F. Weller. 2000. Practical applications of statistical process control [in software development projects]. *IEEE Software* 17, 3 (May 2000), 48–55. DOI: https://doi.org/10.1109/52.896249

[96] Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP'15)*. 1711–1721.

[97] Etienne C. Wenger and William M. Snyder. 2000. Communities of practice: The organizational frontier. *Harvard Business Review* 78, 1 (2000), 139–146.

[98] Terry Winograd, Fernando Flores, and Fernando F Flores. 1986. *Understanding Computers and Cognition: A New Foundation for Design*. Intellect Books.

[99] Jeffrey M. Wooldridge. 2012. *Introductory Econometrics: A Modern Approach* (5th ed.). Cengage Learning.

[100] Judy L. Wynekoop and Diane B. Walz. 2000. Investigating traits of top performing software developers. *Information Technology & People* 13, 3 (2000), 186–195.

[101] Daniel Bärl Torsten Zesch and Iryna Gurevych. 2012. Text reuse detection using a composition of text similarity measures. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING'12)*, Vol. 1. Citeseer, 167–184.

[102] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. 2017. On the scalability of Linux kernel maintainers' work. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*. Association for Computing Machinery, New York, NY, 27–37. DOI: https://doi.org/10.1145/3106237.3106287

Received July 2019; revised February 2020; accepted March 2020