UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Supporting Software Developers with Activity Event Analyses and a Management Console for Automation

Permalink

https://escholarship.org/uc/item/73p9d4rm

Author

Wang, Zhendong

Publication Date

2023

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NoDerivatives License, available at https://creativecommons.org/licenses/by-nd/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Supporting Software Developers with Activity Event Analyses and a Management Console for Automation

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Software Engineering

by

Zhendong Wang

Dissertation Committee:
Professor David F. Redmiles, Chair
Professor Anita Sarma
Associate Professor James A. Jones

2023

© 2023 Zhendong Wang

TABLE OF CONTENTS

	Page
LIST OF FIGURES	**iv**
LIST OF TABLES	**vi**
LIST OF ALGORITHMS	**vii**
ACKNOWLEDGMENTS	**viii**
VITA	**ix**
ABSTRACT OF THE DISSERTATION	**xii**
1 Introduction	**1**
1.1 Dissertation Outlines	5
1.2 Expertise of Software Engineering	6
1.3 OSS Development	8
1.3.1 Hierarchical OSS Community	8
1.3.2 Role-Based Relationships among Developers	10
1.3.3 Contributing Activities of OSS Developers	12
1.3.4 OSS Ecosystems	14
1.4 Small-Scale Automation for SE	15
1.4.1 Automated Workflow with SE Bots	15
1.4.2 Challenges and Opportunities of SE Bots	17
2 Unveiling Elite Developers' Activities	**18**
2.1 Elite Developers in OSS Projects	18
2.1.1 Empirical Study Design on Prevalent OSS Projects	21
2.1.2 Results and Findings	36
2.2 Elite Developers in OSS Ecosystems	51
2.2.1 Empirical Study Design on Major OSS Ecosystems	53
2.2.2 Results and Findings	58
3 Small-scale Automation for Software Development	**67**
3.1 The Usage of SE Bots on GitHub	68
3.1.1 Bot Identification Study Design	70
3.1.2 The State-of-the-Art of SE Bots on GitHub	76

3.1.3 Finding Summary 80
3.2 Bot-Assisted Workflows for Individual Practitioners 82
3.2.1 Research Methods 84
3.2.2 Interview Findings 86

4 Bot Management Console **93**
4.1 Management Console for SE Bots 93
4.2 Design Rationales 96
4.2.1 Design Guidelines and Requirements 96
4.2.2 Architecture Decisions 98
4.3 Principle Use Cases 101
4.4 Evaluation though Simulated Bot Deployment 106
4.4.1 ODD Protocol 109
4.4.2 Analysis Methods 115
4.4.3 Evaluation Results 116

5 Discussion **121**
5.1 Empirical Studies on Elite Developers 121
5.1.1 Contribution Activity and Trend of Elite Developers 121
5.1.2 Impact on OSS Communities and Artifacts 123
5.1.3 Practical Implications 126
5.1.4 Design Implications 128
5.1.5 Recommendations 129
5.2 Empirical Studies on Automated Workflow and SE Bots 131
5.2.1 Increasing and Integrating Applications of SE Bot 131
5.2.2 Developer Experience and Expectation of Future SE Bots 133
5.3 Limitations 135

6 Concluding Remarks and Future Work **138**

Bibliography **141**

Appendix A SE Bot User Study **157**

Appendix B Empirical Data Repository **165**

LIST OF FIGURES

Page

1.1 A synthesized and onion-shaped OSS development community structure summarized by Crowston and Howison [37]. 9
1.2 The basic human-bot interaction workflow described by Liu et al. in their 2020 publication [103] 16

2.1 Data collection and cleanup process. 23
2.2 The taxonomy of GitHub event types. 25
2.3 `PullRequestReview` Events: The overall code review in highlighted part A triggers `PullRequestReview` and specifically to a unified diff in part B triggers `PullRequestReviewComment` Event. 28
2.4 Elite developers' effort allocations in each category. 36
2.5 Comparison of an individual elite and non-elite developer's average activity amount during a project-month. 38
2.6 Trends of individual developer's activities in the four activity categories of `Tensorflow`. 40
2.7 Changes of time-related effects in the regression models. 47
2.8 Data collection and preparation process. 56
2.9 The proportions of elite developers' activities. 60
2.10 The pairwise similarities and differences between the relationships represented by the regression models. 65

3.1 GitHub bot identification study process 71
3.2 Frequency of bot function combinations and overall bot function frequency (in sampled projects) 77

4.1 Architecture of proposed bot management console framework 101
4.2 User Interface for automation selection 102
4.3 UI for controlling notification and access permission 103
4.4 The agent-based system described in this simulation study. The lower part of the figure presents the real-world interactions between actors and the code host environment, and the upper part is the representation and simulation of notifications produced by this agent-based model. In addition, solid lines represent interactions with the environment, and the dashed lines represent communications between agents. 108
4.5 Notifications generated from comments for five sampled repositories 118

4.6 Notifications generated from issue and pull request for five sampled repositories 119
4.7 Repository notification from comments with three combinations of bot deployed 120

LIST OF TABLES

Page

1.1 Comparisons between elite developers and other similar roles defined in the literature. 11

2.1 Sampled projects and their description. 21
2.2 Research questions and corresponding data analysis methods for project study. 32
2.3 Descriptive statistics on activity amount for each sampled project. 37
2.4 Regression models for project productivity. 42
2.5 Regression models for project quality. 46
2.6 List of sampled OSS ecosystems and projects. 54
2.7 Research questions and corresponding data analysis methods for ecosystem study. 55
2.8 Descriptive statistics for elite developers' activities. 59
2.9 Panel regression results. 62

3.1 Top ten of most prevalent bot services among popular OSS development projects 81
3.2 Demographic details of interview participants 86

4.1 Design guidelines of bot management systems 97

LIST OF ALGORITHMS

Page
1 Stale Bot Behavior 112
2 Welcome Bot Behavior 112
3 Coveralls Bot Behavior 113

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my doctoral advisor, Dr. David Redmiles, for his invaluable guidance, unwavering support, and mentorship throughout this journey. Her expertise and insights have been instrumental in shaping this research and beyond, and I am immensely grateful for his patience and dedication.

My sincere thanks also go to my dissertation committee members: Dr. Anita Sarma, and Dr. James Jones. Their rigorous critique, insightful comments, and encouragement have greatly contributed to my personal and professional growth.

I wish to thank the Department of Informatics, especially the administrative staff, for their assistance and for creating a conducive environment for research and learning. My research would not have been possible without the funding provided by the Donald Bren School of ICS.

I am grateful to my friends and fellow graduate students for their friendship, intellectual discussions, and shared experiences, which made this journey more enjoyable and less daunting.

I would also like to acknowledge the researchers and authors whose work has contributed to and informed my study. Their pioneering efforts laid the foundation for my research.

On a personal note, I owe a debt of gratitude to my parents for their love, sacrifice, and constant belief in me. To my partner, Lu He, thank you for your unwavering support, understanding, and companionship during the ups and downs of this journey.

VITA

Zhendong Wang

EDUCATION

Doctor of Philosophy in Software Engineering **2023**
University of California, Irvine *Irvine, California*

Master of Science in Software Engineering **2018**
University of California, Irvine *Irvine, California*

Bachelor of Science in Mathematics and Computer Science **2014**
University of Nebraska, Lincoln *Lincoln, Nebraska*

RESEARCH EXPERIENCE

Software Engineering Researcher **2022–present**
SAP Labs *Newport Beach, California*

Graduate Research Assistant **2018–2022**
University of California, Irvine *Irvine, California*

Undergraduate Research Fellow **2014–2015**
University of Nebraska, Lincoln *Lincoln, Nebraska*

Undergraduate Research Assistant **2013–2014**
University of Nebraska, Lincoln *Lincoln, Nebraska*

TEACHING EXPERIENCE

Teaching Assistant/Course Reader **2017–2019**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Zhendong Wang, Yi-Hung Chou, Kayla Fathi, Tobias Schimmer, Peter Colligan, David Redmiles, and Rafael Prikladnicki. Co-designing for a hybrid workplace experience in software development. IEEE Software, 2022

Zhendong Wang, Yi Wang, and David Redmiles. From specialized mechanics to project butlers: the usage of bots in OSS development. IEEE Software, 2022

Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. Unveiling elite developers' activities in open source projects. ACM Trans. Softw. Eng. Methodol., 29(3), June 2020

REFEREED CONFERENCE PUBLICATIONS

Yi-Hung Chou, Zhendong Wang, Tobias Schimmer, Rafael Prikladnicki, and David Redmiles. Adapting software teams to the new normal: An early case study of transitioning to hybrid work under COVID-19. In Proceedings of the 56th Hawaii International Conference on System Sciences, 2022

Zhendong Wang, Yi Wang, and David Redmiles. Competence-confidence gap: A threat to female developers' contribution on GitHub. In 40th International Conference on Software Engineering 2018, Software Engineering in Society, 2018

REFEREED OTHER PUBLICATIONS

Zhendong Wang. Assisting the elite-driven open source development through activity data. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Doctoral Symposium, pages 1670–1673, 2020

Zhendong Wang, Yang Feng, Yi Wang, James A Jones, and David Redmiles. Elite developers' activities at open source ecosystem level. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, pages 312–313, 2020

PRESENTATIONS

"Assisting the Elite-driven Open Source Development through Activity Data", the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Doctoral Symposium, Online, Nov 2020

"Unveiling Elite Developers' Activities in Open Source Projects", Southern California Software Engineering Symposium (SuCSES), University of California, Irvine, CA, Jun 2019

"Expertise Location in Software Engineering", ISR Research Forum, University of California, Irvine, CA, Jun 2018

“Competence-confidence Gap: A Threat to Female Developer’s Contribution on Github”, Gothenburg, Sweden, May 2018

“Visual Resume: hiring in the global stage using card-based developer profiles of online contribution”, ISR Research Forum, University of California, Irvine, CA, Jun 2017

ABSTRACT OF THE DISSERTATION

Supporting Software Developers with Activity Event Analyses and a Management Console for Automation

By

Zhendong Wang

Doctor of Philosophy in Software Engineering

University of California, Irvine, 2023

Professor David F. Redmiles, Chair

By the late 80s, empirical research in various areas of Software Engineering has demonstrated the crucial roles of expert developers in engineering productivity. With recent advances in data digitization, computational power, and automated engineering processes, experts have unprecedented opportunities to leverage their expertise and amplify their impact on the outcomes of software projects. Recent research efforts are determined to identify, categorize and measure these critical human resources to make software engineering processes more efficient while also improving practitioners' well-being simultaneously.

This dissertation advances our knowledge about the productivity factors relating to workflows of expert developer groups mainly in Open Source Software (OSS) and proposes a novel management framework to address major usability, customizability, and extensibility issues in applying small-scale automation. Through three main studies, this dissertation employed multiple methods to ensure its research resilience, including reviewing the literature, mining software repositories, testing statistical hypotheses, interviewing prospective users, and prototyping. In the first empirical study, I coherently grouped atomic platform event logs into sensible developer effort trends and associated elite developers' activities with the outcomes of OSS projects and mainstream ecosystems. Its results suggested that as contributor communities evolve, increasing responsibility for non-coding tasks had a negative association with the technical outcomes of OSS projects. The second study investigated how practitioners automated the accumulating non-coding and repetitive tasks with small-scale automation, SE bots. Through a tiered and mixed-method approach, this study systematically identified and categorized the state-of-the-art SE bots that are prevalent in OSS practice from the top 1,000 popular repositories. Further, I leveraged semi-structured interviews with elite developers to confirm and refine their usability issues and expectations of SE bots. The final study summarizes design guidelines and provides a prototype for a Bot Management Console, which intends to address main usability issues when deploying and applying SE bots from daily practice. The practical effectiveness of this console has been validated by simulated deployments in real-world scenarios.

The main contributions of this dissertation include theoretical development on OSS literature, actionable recommendations for software practitioners, and practical implications on bot design and implementations. The reported results also provide future research directions for improving expert developers' well-being, engineering productivity, and ultimately the sustainability of OSS communities.

Chapter 1

Introduction

Empirical research has demonstrated the crucial roles of expert developers in engineering software products for the past decades [49, 119, 139]. With advances in data digitization, computational power, and engineering process, these experts have unprecedented access to amplify their impact on software project outcomes and other peer practitioners in the software industry [172, 173].

Compared to the conventional software development process, modern collaborative software development paradigms, such as the Open Source Software (OSS) and Globally-distributed Software Development (GSD) models, provide practitioners with the extensive availability of manpower across the globe. However, these distributed models require building teams virtually and establishing labor divisions with limited developer profile information in advance [37, 73, 118, 119]. This recurrent downside of these models is also the obstacle of locating expertise in its essence [112]. Moreover, practical but oversimplified mechanisms of task delegation and expertise location have overburdened groups of central individuals as these problems escalate [112]. These risks foreshadow potential productivity loss, and neglecting expert developers' cumulative burden could sabotage their work experience substantially.

Many recent research efforts seek to identify, categorize, measure, and support this critical human resources [14, 151, 166, 172]. Substantial software engineering research designed and developed novel tools and improved engineering processes to enhance experts' Developer Experience (DX), with the goals of improving productivity, satisfaction, engagement, and retention [67]. While other studies made tremendous progress in this stream of research, in this dissertation, I argue that incrementally enhancing the current practice of expert software developers is a justifiable and effective way to improve their work experience [28, 67], other than introducing novel technologies or adapting to advanced engineering processes drastically.

Elite developers serve as the crucial pillars underpinning the sustainable maintenance of OSS projects. However, as these projects and their contributor communities expand, the augmented workload related to non-coding tasks exposes the risk of adversely affecting their technical contributions. Thus, this dissertation proposes that the engineering workflow of elite and expert software developers could be supported with an automation management console that is capable of integrating current atomic automation features and controlling excessive interruptions and noises. Further, the console should also provide extensibility for features of social and technical values in the collaborative development process. To summarize:

This thesis seeks to,

- expand our understanding of elite developers by examining how their activities evolve and impact the technical outcomes of OSS projects;
- investigate the critical roles of SE bots in OSS projects and their influence on the workflow of elite developers, and identify their usability challenges;
- propose an automation management console that integrates current atomic automation features and reduces noise to optimize workflow efficiency.

Therefore, the research agenda of this dissertation is as follows: 1) investigating the current challenges of expert software developers empirically, especially ones regarding their technical and non-technical workflow; 2) unpacking and understanding their current practice for addressing these challenges with small-scale automation, e.g., SE bots; 3) identifying limitations of current SE bots, and prototyping demonstrations aiming for gradual improvements.

For investigating this overarching thesis statement, this dissertation employs multiple research methods through three major studies. Primary research methods in this dissertation include literature reviews, inferential statistics, qualitative inquiries, and user studies with practitioners. The first empirical study aimed to comprehensively understand elite developers' on-platform activity and their impact on OSS projects and ecosystems. As a mixed methods research study, it leveraged enormous categories and amounts of software engineering data, including raw platform log event, commit messages, issue/PR reports, and so on. This study employed inferential statistics, e.g., ANOVA tests and panel regression, and identified a predictive relationship between developers' activity trends and project outcomes, especially along with the community evolving. The second study sought to investigate expert developers' workflows of non-coding activities with automation assistance. This study employed a state-of-the-art bot classifier to identify an initial set of prevalent bot services on

GitHub, and further manually validated and identified additional bots through a multi-tier identification process. By collecting the usage data and attributes of these bots, I was able to compile a list of popular bot services on this platform. Moreover, I conduct interviews with expert developers about their challenges of integrating bots into their workflow, and their practical expectations of future SE bots. Finally, the last study summarizes the design guidelines and requirements for a management console of SE bots. The console's designs were evaluated by simulated deployment, supported by OCD protocol of multi-agent simulation.

The findings of this dissertation have both practical and research value to the software engineering community. By categorizing elite developers' activities in OSS projects and broader ecosystems, there was a significant trend of investing more time into non-coding activities as the project developed and its community evolved over time, which had worsened the technical outcomes of OSS projects [171, 172]. Hence, the second study investigated the usage of SE bots in OSS communities, as SE bots were one of the latest practical efforts for alleviating these supportive and organizational burdens. The empirical results suggested that these SE bots have been widely adopted and become repositories' "butlers" who performed repetitive housekeeping tasks of these repositories [174]. Furthermore, through interview studies, practitioners were expecting more technical and social values in future SE bots and also expressed concerns about bots' excessive notifications and difficulties in integrating their functionalities. Therefore, in the third study, I designed and prototyped a *Bot Management Console* that particularly aimed to solve the notification and functionalities integration problems of current SE bots. This last study summarized six major guidelines for designing SE bot consoles from the above studies and SE bot literature, and further presented the designs and implementations of this console. The console's evaluation results suggested a substantial improvement in the usability of bot services.

To briefly summarize, the main contributions of this dissertation include the following: first, to empirically identify how expert and elite developers behave and influence software projects and developer communities; second, to advance our knowledge of OSS developers' usages, challenges, and expectations of SE bots; and third, to propose an advanced automation management console which assists elite developers by managing their bot-assisted workflow. With such an advanced management console, this dissertation provides the potential to improve developers' well-being, i.e., "how healthy and happy they are, and how their OSS work impacts it" [59]. Ultimately this work has the potential to improve the productivity and sustainability of OSS communities.

1.1 Dissertation Outlines

This dissertation is organized as follows. The remainder of this first chapter offers an in-depth literature review, encompassing the key foundations upon which this research is built: the expertise of software developers, the evolution of OS communities, and the emerging role of SE bots.

Chapter 2 presents empirical studies conducted within OSS projects and ecosystems. This includes a detailed overview of the study design, a comprehensive presentation of the results, and a discussion of their implications. The principal findings of this chapter have been published in the Transactions on Software Engineering and Methodology (TOSEM) and in the Proceedings of the 42nd International Conference on Software Engineering, Companion Volume (ICSE 2020 Companion).

Chapter 3 begins by exploring the prevalence of SE bots in some of the most successful repositories on GitHub. It then provides insights from interview studies conducted with expert developers, focusing on their daily workflows, particularly regarding their interactions with SE bots.

In Chapter 4, I introduce the design guidelines for a management console for SE bots. This chapter features my designs and implementations of the console, complemented by a simulation evaluation. The initial concept of the management console was first published in the Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020) Doctoral Symposium. The bot usage study has also been featured in IEEE Software 2022 and in the Proceedings of the 45th International Conference on Software Engineering, Companion Volume, BotSE workshop (ICSE 2023 Companion).

Chapter 5 discusses the empirical findings and implications of three studies, and finally, Chapter 6 concludes this dissertation with several major research directions for the future.

1.2 Expertise of Software Engineering

Software engineering is a human-centered activity that requires qualified practitioners with proper expertise [25]. Effectively managing expertise in a software organization can benefit a broad spectrum of its routine activities with improved productivity, such as workforce hiring and training, maintaining project development, and coordinating among teammates [14]. Furthermore, fostering an organization's expertise network improves its technical and social outcomes [186]. However, how to effectively locate expertise in the broad context of software development remains a critical challenge in software engineering research and has been an ongoing investigation.

Emerging software development paradigms such as large-scale GSD and the voluntary contribution model in OSS development blur the boundaries of traditional software engineering organizations. These advances result in more difficult challenges and higher demands for locating expertise during software development [73, 124]. However, while the importance of developer expertise has been unanimously acknowledged in the software engineering community, we have not yet developed a comprehensive understanding of its characteristics. Furthermore, it remains unclear how the state-of-the-art expertise location techniques utilize these characteristics, along with the unprecedented amount of data preserved by modern collaborative development tools [18, 42]. Lacking such knowledge impairs the ability of researchers and practitioners to design, implement, and adopt effective expertise location tools. For instance, prior studies have repeatedly reported that practitioners resort to simple approaches to locate expertise, such as asking the "broker" member of colleagues and applying the "line-10" rules on the code artifact, rather than using sophisticated expertise locating tools [112, 113, 185].

Every organization hopes to leverage its existing expertise resource collectively and effectively, but this goal is often hard to reach, especially for large-scale organizations with complex structures and a diverse pool of talents. For a software development organization, failing to identify experts is a threat to overall productivity, potentially overburdening a few central individuals [113]. Thus, identifying specific domain expertise from individual talent within the organization is critical for its outcomes.

Throughout the past decades, various studies have researched how to identify expert behaviors and locate experts. As mentioned in earlier sections, Brooks argued that "good designers" had a significant positive impact on the quality and productivity of software products [25]. Moreover, these expert developers not only boost project outcomes but also improve other team members' performance. For example, Sonnentag found that the "excellent" group of developers usually spends more effort in consultation meetings and supporting other developers [151]. More recently, Petre and Van der Hoek summarized three key attributes of expert developers in the modern age [128]. First, expert developers "know their stuff." Besides knowing the basic fundamentals, they usually develop a comprehensive understanding of their knowledge field, including awareness of what they do not know. Second, they are normally social designers, bringing the best possible developers together. In the OSS context, core developers usually can be found at the center of the project's social network according to Joblin et al [81]. Finally, Petre and Van der Hoek point out that expert developers practice the "designly" thinking, i.e., they have a unique mode of problem-solving, such as focusing on the essence of the problem, proposing alternative solutions, and more. This argument also aligns with earlier findings about the knowledge representations of expert v.s. novice developers [114]. However, the expert characteristics summarized by the software engineering literature are often intangible and hard to articulate or quantify.

1.3 OSS Development

Open source software (OSS) has become an engine for innovation and critical infrastructure for modern software development [39]. OSS development is often supported by communities formed from a loose collection of individuals. The contribution from these individual developers consists of various SE 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 collectively influence their outcomes.

1.3.1 Hierarchical OSS Community

OSS development has been a mainstream practice in building modern software systems [39]. Different from traditional software development paradigms, an OSS project is centralized in its community, which produces collective public goods through collaboration among its members [76]. Though the detailed governance and social practices may vary across different projects [125], members of an OSS community usually have different roles, which entail different responsibilities, permissions, and levels of contributions [17, 141]. Similar to other hierarchical organizations, an OSS community follows an onion-shaped social structure [37]

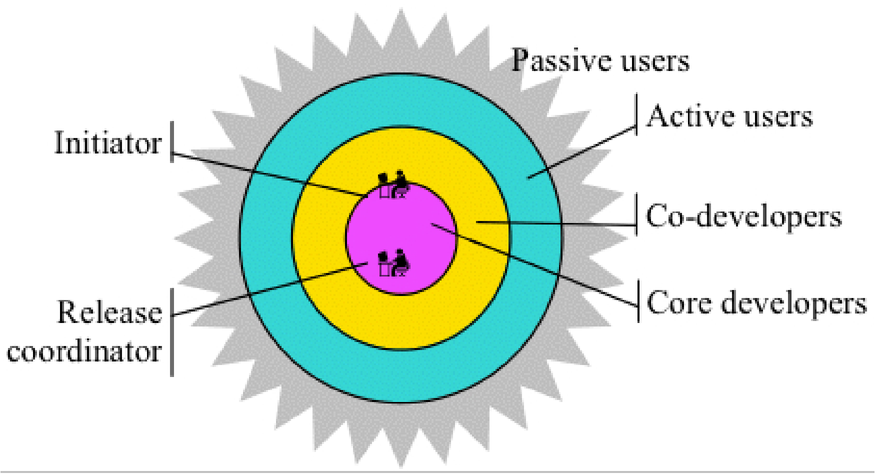


Figure 1.1: A synthesized and onion-shaped OSS development community structure summarized by Crowston and Howison [37].

(see Figure 1.1).

There are several different definitions for each layer in this hierarchical community [37, 49, 79], 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 outmost layer to peripheral members (note that terms may differ from study to study). Especially, each member of a project may play several roles. Peripheral members of an OSS project typically are users of the software artifacts and products 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 they have achieved recognition in the same software ecosystem and developer community [80]. If these peripheral members wished to contribute to more critical tasks of the project, they usually have to undergo a socialization process. In Ducheneaut's case study, he unveils a socialization path of becoming a core Python developer which started as a periphery user [49]. This path included socializing with the current core development team and completing a series of development tasks that increased in difficulty. After being socially recognized and technically acknowledged by existing experts for a project, they joined the core team to become one of the core developers and gain the privileges of this project (i.e., "tenured" in a repository). Further, they begin to hold administrative power in the project; for example, they were responsible for conducting quality assessments on other external contributors' technical submissions.

Expanding the socialization process to a larger community is becoming more common in current development practices, particularly for large-scale projects. One way to describe this approach is as an "ecosystem" umbrella, where software engineering and collaborative techniques enable more projects to be developed in parallel with similar domain knowledge, technical stacks, and community norms. Jergensen et al. discussed how recognized OSS experts transfer their expertise and reputations to other similar OSS projects, highlighting the evolution of this socialization process in modern OSS development [80]. In a case study of Linux, one of the largest OSS ecosystems, Iannacci found that OSS users contribute to different projects in various ways, with many developers moving from project to project like "nomads" [77].

Although modern OSS are often developed by large communities, research indicates that only a small portion of developers contribute to the majority of the codebase [37, 79, 96]. Understanding the activities of these "elite" or "core" developers is crucial for assessing the health and sustainability of OSS communities. To this end, various methods have been employed to analyze their contributions. In mature and stable OSS projects that follow the "bazaar" style, as described in Raymond's ideology [131], members have developed shared authorities and privileges, as well as mechanisms for transferring them among each other. As a result, a member's status as an "elite" or "core" developer is dynamic [125].

1.3.2 Role-Based Relationships among Developers

Developers in a software project have complex relationships with each other, engaging in various development activities. While "core vs. peripheral" is a dominant terminology used in SE literature to characterize role-based developers' relationships, other terminologies have been proposed. For example, when newcomers or outside developers join an existing project, a *mentor-newcomer* relationship may be established for social and technical reasons [29, 43]. However, previous research suggests that mentorship activities face various barriers for both mentors and newcomers, which may threaten project sustainability [11]. Additionally, understanding the contribution behaviors and impacts of *internal* and *external* contributors in company-sponsored projects is critical. For instance, companies need to balance management efforts and fast iteration of enhancements when receiving external help. As mentioned earlier subsection, developers' roles change over time, and role migrations are common [80]. A peripheral member may be promoted to a core member, and a newcomer may become a mentor after their skills develop. However, such a growing process may take substantial effort, as external members may need to undergo a time-consuming process to gain the roles of internal members [47, 49]. Furthermore, role-based relationships are dynamic in nature, as core members may lose their roles if they no longer actively contribute to the project [102]. Therefore, role-based relationships are dynamic in nature. Table 1.1 provides an overview of studies investigating the relationships between developers, highlighting the complexity of role-based relationships in software development projects.

Table 1.1: Comparisons between elite developers and other similar roles defined in the 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 [119].
Maintainer	Contributor	Open source	Members who are responsible for a software module, mainly in accepting contributed patches [49, 79, 189].
Internal	External	Company-sponsored Open source	Individuals who are members of the development group; usually are listed as contributors on the project homepage [47, 169].
Mentor	Newcomer	General software development	Persons who train and help novice and inexperienced developers (newcomers) for their onboarding. [11, 29, 43].
Elite	Non-elite	Open source	Developers who own administrative privileges in the project [this study].

Influenced by the dynamics of development roles, OSS 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 [42, 49]. Meanwhile, supported by version-control systems and logs from various communication channels, a tremendous volume of social and technical latent data can be acquired [16]. 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 gain insights into the socio-technical processes of an open-source project and improve them, it is crucial to obtain information about role-based relationships [81]. Such information can be utilized to extract developers' expertise [12, 120], model a project's lifecycle [82], or predict project outcomes [30, 166]. However, to achieve these utilities, it is necessary to identify role-based relationships from multiple perspectives. Typically, researchers rely on count-based measurement for technical contributions to identify role-based relationships. This approach assumes that "core" developers perform more activities than others do. However, this perspective may be insufficient for some scenarios. Recently, Joblin et al. proposed a network-based perspective for identifying core and peripheral developers, which captures structural roles within social and technical dependencies beyond counting technical contributions. This approach suggests that researchers need to adopt fresh perspectives in recognizing developers' roles that fit their specific research scenarios.

1.3.3 Contributing Activities of OSS Developers

Several studies have investigated the activities and tasks of developers in software engineering research [37, 49, 189], providing critical insights into how project members collectively deliver software systems in complex contexts. For example, Baltes and Diehl developed a conceptual theory to characterize the complex relationships among developers' activities, the formation of expertise, and performance by surveying 335 software developers and literature on expertise and expert performance [12].

To derive actionable insights for software development, research efforts have attempted to categorize diverse activities into a few high-level sense-making categories. Sonnentag conducted an empirical study with software-company professionals to study their weekly activities in software development [150]. 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*. Her study is critical in identifying and classifying the full spectrum of software engineers' activities, which serves as theoretical foundations for the analysis of our work. However, there is still a need to develop a mapping between fine-grained raw events in open source development and these high-level categories.

With a baseline of activity categorization, studying developers' workload becomes necessary as performing technical and non-technical activities is not effortless. Zhou et al. highlighted that the workload per maintainer varies according to modules, which forms different balances to make these modules sustainable [189]. 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, with a small number of developers often bearing 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 [157].

Roles in open-source projects also determine members' different activity focuses and work efforts [92]. Nevertheless, in software engineering literature, the complex division of labor in an open-source project has not been well studied, particularly with explicit consideration of role-based relationships and different types of activities beyond technical contributions. Our "elite vs. non-elite" terminology is designed to capture this often-neglected perspective, with the "elite" obtaining managerial responsibilities in an open-source project. This perspective, combined with recognizing different types of activities, including both technical contributions and non-technical efforts, can offer valuable insights regarding developer roles that may not be captured by other existing perspectives and corresponding operationalizations.

1.3.4 OSS Ecosystems

The ecosystem model has been widely adopted in OSS development. It creates value by integrating software projects related to a given domain and attracts many contributors, as well as users, to maintain and enhance its community. Since the mid-2000s, researchers have devoted substantial efforts to analyze and improve the ecosystem model from various perspectives [80, 87, 165]. SE researchers have extensively analyzed OSS development processes [142], evolution models [82, 161], and social structures [44, 49, 170]. All are considered to be strongly correlated with the sustainability and outcome of OSS projects developed under ecosystems. Schultis et al. conducted an empirical study to analyze the architecture and collaboration process in two ecosystems [144]. They summarized their organizational structures and collaboration processes into three models, and identify multiple architectural challenges in the ecosystem. German et al. explored the contribution characteristics of the R software ecosystem and their findings highlight that user-contributed packages have much less source code and documentation and also receive less attention than core packages do [62]. Crowston et al. reviewed the literature on OSS development and summarize issues related to processes, emerging states, and outputs of OSS ecosystems [39]. Moreover, as more OSS projects are commercially involved, Zhang et al. found that though company contributors engage in the open source ecosystem with various strategies, their participation has a positive association with the ecosystem productivity [188].

1.4 Small-Scale Automation for SE

1.4.1 Automated Workflow with SE Bots

With the increasing popularity of the software ecosystem paradigm and the complicity of interactions between various stakeholders, there is a demand to automate or just semi-automate various technical and social tasks in software development. In the past decade, the emerging CI/CD model and the large-scale collaborative software development powered by online social coding platforms have driven the rapid evolution of OSS development practices. The fast iterations drastically complicate the development process as projects evolve. Therefore, practitioners resolve to exploit automation techniques that handle specific routine development and social tasks, so they could focus on innovative activities.

These automation techniques, i.e., bots, may support developers through many service mediums. Some bots have user profiles and are authorized as their projects' contributors [180]. Besides these account-based bots, others may employ platform applications, user settings, and various external services [103]. Practitioners often adopt desired bot services with trade-offs between performance, privacy, and security.

The prevalence of automation technology in the form of SE bots has made them a crucial component of daily engineering processes within many OSS organizations, serving as the primary interface for human-AI interactions [103](see Figure 1.2). They serve as an extension of the development team, i.e., repository *butler*, providing automated feedback and monitoring of code quality, release management, and community engagement [97, 174]. With their ability to perform repetitive tasks and improve efficiency, SE bots have enabled developers to focus on more creative and complex tasks. Overall, SE bots have significantly impacted the way OSS organizations operate, making their daily engineering processes more streamlined [51].

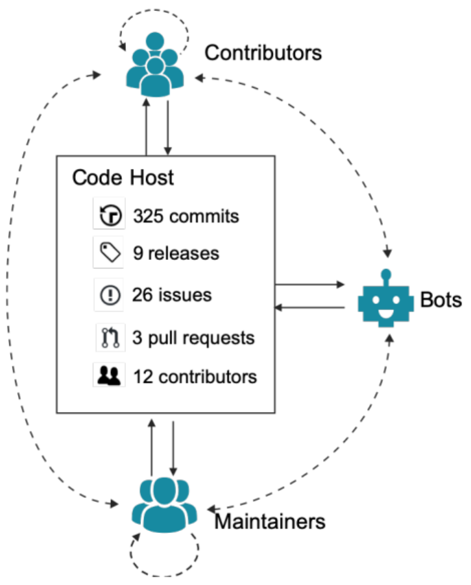


Figure 1.2: The basic human-bot interaction workflow described by Liu et al. in their 2020 publication [103]

1.4.2 Challenges and Opportunities of SE Bots

Despite the widespread adoption of SE bots in OSS, there are still challenges and limitations that need to be addressed. Previous studies have employed various methods, such as surveys, interviews, and empirical analyses, to identify these issues and propose solutions. Frist, one major issue is the limited understanding of user needs and preferences, which has resulted in significant usability problems [52, 103, 179]. Second, without clear standardization, practitioners have difficulties trusting automated decision-making processes [27, 52]. Third, the evolving nature of software technology poses challenges in maintaining and updating bots, and they are often slow to keep up [51, 174]. To address these issues, several bot development frameworks have been introduced with the aim of providing a standardized and transparent bot creation process, including OpenBot, Probot, Octokit, and others. Additionally, Wessel et al. have provided general guidelines for developing bots for GitHub, with a focus on supporting social coding platforms [181].

Meanwhile, researchers have collected practitioners' feedback for future improvement. For instance, they find practitioners expect more *smart* features in bots' communication and functionality [180]. Erlenhov et al. also emphasize the importance of a bot being socially competent and bringing technical value, especially when they are designed for *smarter* and more complex tasks [51]. Liu et al.'s survey created seven principles for designing bots that demonstrate the critical challenges in interactions [103].

Chapter 2

Unveiling Elite Developers' Activities

This chapter introduces two studies that intend to empirically investigate how expert developers act and make impacts on OSS projects and ecosystems. Particularly, these studies leverage a notion of *elite developers* to represent these socially recognized experts in OSS projects [49], and therefore they were identified by their permissions and privileges in their software repositories and organizations. The first study investigates 20 large standalone OSS projects, and the second study follows a similar methodology but extends the scope to five major OSS ecosystems.

2.1 Elite Developers in OSS Projects

As mentioned earlier in the introduction, OSS has become an engine for innovation and critical infrastructure for modern software development [39]. Its 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.

Software developers maintain diverse activity profiles, including implementing new features, documenting changes and design, analyzing requirements, and fixing bugs [96]. Contributing source code is only one type of the activities pursued by an elite developer. Prior studies typically provide insight into one such specific non-coding activity, e.g., peer review [133] or committing code [41]. 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 [70, 147] and estimating cost [6]; 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 also non-technical 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 expertise from the community. This leads to the first research question:

$\mathbf{RQ_{2.1}}$ *What do elite developers do in addition to contributing typical technical code in OSS projects?*

Since software engineering is a human-centered activity [57], effectively managing its human resources may significantly enhance project productivity and collaboration quality. However, it is not clear what kind 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 also can assist resource management. This gives rise to the second research question:

RQ$_{2.2}$ *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 activities require qualified developers with the proper expertise to complete the task efficiently and effectively, understanding these impacts are critical for developers to oversee the project and assure productivity and product quality. Thus, we have our third research question:

RQ$_{2.3}$ *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, this chapter conducts an empirical study using fine-grained event data from 20 large OSS 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, this study first maps 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 2.1.1. This study then uses multiple techniques to model and analyze the data.

This study reveals three main findings. First, elite developers participate in a variety of activities, of which coding only accounts for a small proportion. Second, with the progress of the project, elite developers tend to be increasingly involved in more non-technical 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 (that is, the number of new commits and average bug cycle time in each project-month), the results suggest that project productivity has negative correlations with non-technical (communicative, organizational, and supportive) activities. For one project quality indicator (number of new bugs in each project-month), the results also show that project quality has negative correlations with efforts in non-technical activities; however, for the other project quality indicator (bug fix rate in each project-month), the results suggest that project quality has positive correlations with supportive activities.

2.1.1 Empirical Study Design on Prevalent OSS Projects

To answer the three research questions presented, I conduct an empirical study on 20 selected large-scale OSS projects. This section introduces the design of the study.

Table 2.1: 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.

Targeted Projects

This study selected 20 open source projects hosting their repositories on GitHub as the targets of the study. Table 2.1 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`), a multi-media 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 [169]. Last, the sampled projects should maintain relatively long traceable records on GitHub, which allows us to study the longitudinal dynamics while maintaining data consistency.

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[1]. 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 2.1 provides an overview of the data collection and cleaning

[1]GitHub Event API: `https://developer.github.com/v3/activity/events/`

process.

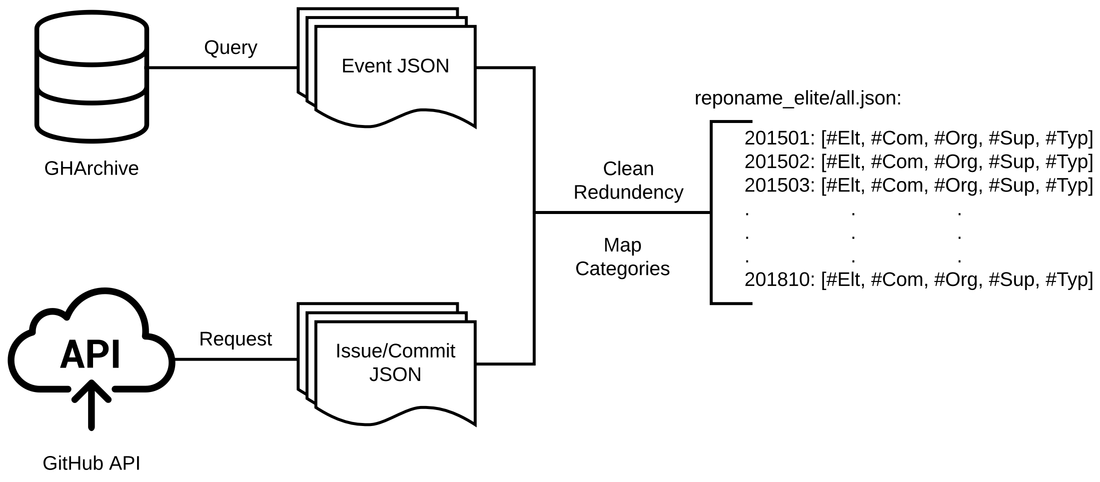

Query
Event JSON
GHArchive
reponame_elite/all.json:
201501: [#Elt, #Com, #Org, #Sup, #Typ]
201502: [#Elt, #Com, #Org, #Sup, #Typ]
201503: [#Elt, #Com, #Org, #Sup, #Typ]
201810: [#Elt, #Com, #Org, #Sup, #Typ]
Clean Redundency
Map Categories
API
Request
Issue/Commit JSON
GitHub API

Figure 2.1: Data collection and cleanup 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. This study uses SQL-like queries (designed by BigQuery) to search for projects and save the results into tables of the personal Google Cloud database. Further, the query scripts exported tables as JSON files to the cloud storage and may download to a local computer for later analyses. In total, this data set comprises 5.60 GB 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[2] library to request events from the GitHub API, and then to download the 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

[2]Simplified HTTP request client for Python: `https://github.com/request/request`

easily search for commits by date and author, and derive necessary metrics for the later data analysis on the project productivity, we also download the commit logs of all sampled projects. In total, 1.81 GB of data on issue events and commit logs were collected.

Finally, some Python scripts were created 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 recorded 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, this study keeps events from the GitHub Issue API which were real-time archives. Then I convert event logs into a monthly list based on the number of events that occurred in each major category.

Event Categories and Mapping

Although the event log data faithfully records developers' activities, there is still a need to recode the data unit categories that are easier for humans to interpret and conduct further analyses. Particularly, the percentage of types for a developer group should be able to reflect the trend of effort allocation and suggest their 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 [24, 150, 151, 184]. One prior field interview study focuses on the daily activities category of professional software developers, which were summarized based on subjects' reported activities [24]. Since this study is particularly interested in investigating the overall activity profile of elite developers, I 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.

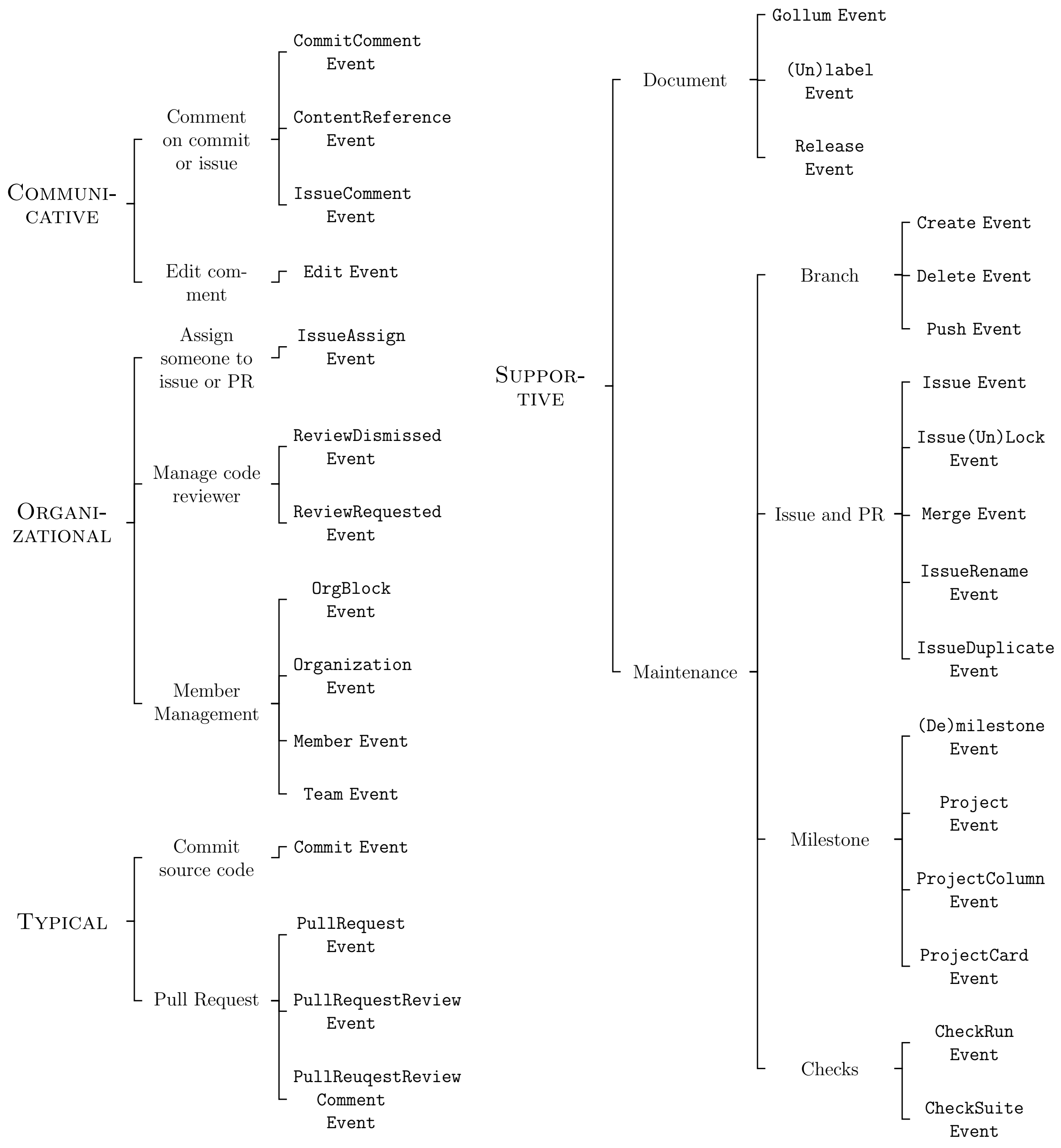


Figure 2.2: The taxonomy of GITHUB event types.

This study reuses an activity categorizing system created by Sonnentag [150] 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, the research subjects were developers in private companies; to adapt these definitions to OSS development, we slightly modify the definition and operationalization of each category as follows:

Communicative: In the conventional co-located software development team, communicative activities usually refer to formal and informal meetings and consultations [150]. 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 [16]. Moreover, some projects such as Tensorflow apply additional broadcast channels, such as a blog, website, and YouTube channel. Therefore, similar to other empirical studies with OSS developers, we are not able to collect communicative activities of 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.*

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.

The definition of organizational activities is *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.*

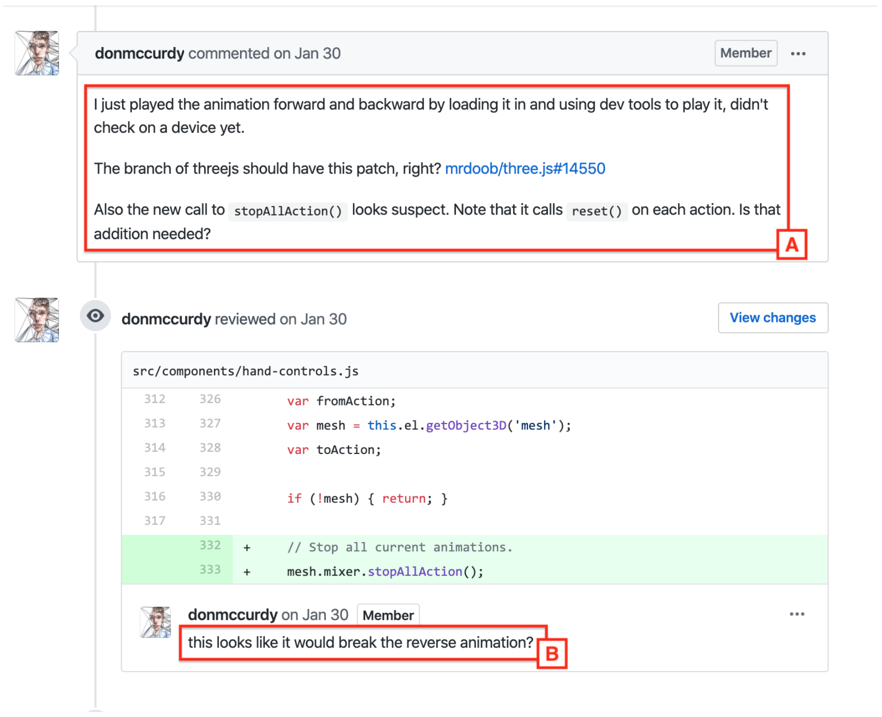
Supportive: Supportive activities are critical to OSS development and mainly refer to other non-coding activities in collaboration. It includes *documentation* work such as writing documentation/wiki-page and categorizing issues by adding labels to them. Further, supportive also includes *maintenance* work, for example, managing development branches and releasing or archiving code versions.

The definition of supportive activities is *non-coding activities in the collaborative open source development that are performed through techniques supported by the platform, including documentation, versioning control, and development branch management.*

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

The definition of typical activities is *conventional code-writing task finished at the individual level, and counted as submitted code reviews, commits, and pull requests.*

Mapping raw events to the above categories: Then I apply the closed card sorting method to place 35 raw GitHub events into the above four major categories. With the help of other two software engineering researchers, we performed 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 Fleiss' $\kappa = 0.77$. All these metrics indicate satisfying inter-rater reliability, and all differences among card sorters are discussed and resolved. The final mapping is shown in Figure 2.2.


donmccurdy commented on Jan 30

Member

I just played the animation forward and backward by loading it in and using dev tools to play it, didn't check on a device yet.

The branch of threejs should have this patch, right? mrdoob/three.js#14550

Also the new call to `stopAllAction()` looks suspect. Note that it calls `reset()` on each action. Is that addition needed?

A

donmccurdy reviewed on Jan 30

View changes

src/components/hand-controls.js

```
       var fromAction;
       var mesh = this.el.getObject3D('mesh');
       var toAction;

       if (!mesh) { return; }

+      // Stop all current animations.
+      mesh.mixer.stopAllAction();
```

donmccurdy on Jan 30 Member

this looks like it would break the reverse animation?

B

Figure 2.3: `PullRequestReview` Events: The overall code review in highlighted part A triggers `PullRequestReview` and specifically to a unified diff in part B triggers `PullRequestReviewComment` Event.

In the aforementioned card sorting and disagreement-resolving process, the context of a specific type of event in making classification decisions was heavily considered rather than solely relying on its literal name. For example, there are three major events on GitHub contains 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[3], thus we classify them as communicative activities. They almost never lead to direct changes to a project's code base. However, the event *PullRequestReview-Comment* 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

[3]According to Chris Wanstrath, a 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/`.

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 2.3 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 2.3 as an example), it shall be classified as a type of typical technical work.

By mapping the low-level raw GitHub events into these four categories, we can reason developers' activities at the level that makes sense to understand them as real work practices and human efforts in organizational settings [126], instead of losing in millions of atomic events which are not considered as integrated work practices. This study argues that such a categorizing system precisely and comprehensively reflects the general work practices of professional software developers. The categorization system is mainly inherited from the literature of empirical field observations and interviews with a large number of software development projects and hundreds of professional developers [24, 150]. Although this study focuses on OSS 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 [36, 66, 137, 168]. Doing so enables us to better study the dynamics of elite developers' work practices and their impacts, thus deriving meaningful findings and implications.

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 ($\mathbf{RQ_{2.3}}$), project outcomes data is also collected. This study considers and models two perspectives of project outcomes: productivity and quality, which are viewed as the most important [166]. Each of them has two indicators as proxies, and they are introduced as follows.

For productivity, the first indicator is that *the number of a project's new commits in a project-month*[4]. Thus, for project i in month m, this study uses $NewC_{im}$ to denote it. In many studies focusing on the OSS development and community, the number of commits is considered as the productivity metric [163, 164, 166]. This study first adopts this widely used productivity indicator for simplicity. Note that this study counts the commits from all contributors rather than from elite developers only because it intends to measure the impact on the productivity of the whole project 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, this study uses BCT_{im} to denote it. Such an indicator has been used to measure project productivity in many prior studies, especially for collaboratively software development [89].

Following the conventions in prior literature [86, 130, 166], this study first operationalizes the code quality by *the number of bugs found during a project-month.* This study simply uses $NewB_{im}$ to denote it. On GitHub, the issue can be of various types, e.g., discussion, new feature requests, improvement requests, and so on. To categorize these issues, software developers often employ keywords or labels to tag them. However, because tagging is often project-specific, we adopt Vasilescu et al.'s method to distinguish bug issues from other issue types in this study [166]. Following this prior study, this study sets up a list of bug-related keywords, including *defect, error, bug, issue, mistake, incorrect, fault, and flaw*, and then searches for these words in both the issue tags and issue titles. If any tags or title of an issue contains at least one keyword, then the issue would be identified as a bug issue. Similarly, as the productivity data, this study computes 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 (BFR_{im}), which is defined as:

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

[4]To simply the following discussion, this study employs the term "project-month" to denote *a given month in a project.*

The Bug Fix Rate is one of the key metrics related to the defect removal process [48, 176]. 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 [100]. If $BFR_{im} < 1$, it indicates that the project's quality risk is accumulating.

Identifying Elite Developers

Following the method used in Hanisch et al.'s study [71], I 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 the 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 the write permission of the repository, the developer can perform several types of administrative work and organizational tasks, 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 loses the issue's public attention.

Since GitHub does not allow anyone other than the repository owners to access the list of members obtaining specific permissions, this study applies a permissions check mechanism to determine the elites: when a developer in the repository performs a task that requires the write permission, we tag this developer with "elite-ship" of the repository. According to observation on this data set, I also found that some projects' elite developers might also suffer a survival issue [102], thus this study sets a 90-day[5] as the length of the "elite-ship", and use this time-window to filter developers who were inactive. During this three-month period, if

[5]Literature on survival analysis of open source developer usually use 30 days or 90 days as a time window [102]. When we examined the raw data, we found that 30 days are too short, while 90 days are a moderate time interval to avoid rushing a decisions on determining whether someone had gained or lost the elite identity.

this developer performs any task that also requires write permission, her "elite-ship" would be renewed for another three months, starting from the month when she performed the task.

Compared with other elite-developer-identification methods based on metrics or networks [81], this method has several advantages. First, this 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[6]. Secondly, this method reflects the socialization process of gaining power and status in a community [49]. Thirdly, this method respects the fact that some developers may be nominated as elite developers before making substantial contributions, particularly in recent company-sponsored projects [169]. Lastly, this method avoids dealing with the marginal cases resulting from the intentionally set threshold, e.g., an $\frac{1}{3}$ cut-off [40].

Data Analysis

Table 2.2 presents the mapping between **RQs** and corresponding main data-analysis methods. This sub-section will introduce them in detail.

Table 2.2: Research questions and corresponding data analysis methods for project study.

RQs	**Data Analysis Methods**
$\mathbf{RQ_{2.1}}$	Descriptive statistics
$\mathbf{RQ_{2.2}}$	Descriptive statistics, one-way ANOVA
$\mathbf{RQ_{2.3}}$	Project-specific fixed effects Panel Regressions (LSDV estimator with Diagnostics)

For $\mathbf{RQ_{2.1}}$ and $\mathbf{RQ_{2.2}}$, we only need to apply simple statistical methods to answer them. Answering $\mathbf{RQ_{2.3}}$ requires establishing correlations between effort allocations and project outcomes. This study applies panel regressions, which is a type of econometric technique, to build linear models. Because our data is panel data in its nature: cross-sectional (multiple

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

projects), longitudinal (multiple time window for a specific project), ordinary multivariate regressions such as OLS can not deal with such type of data [183]. 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[7] with many confounding factors when we do not aim for causality (see Wooldridge [183]: Chapter 7, pp. 256; Chapter 14).

All statistical analyses in this dissertation are performed with R 3.4.1 [129], and its associated packages for macOS High Sierra (version 10.13.1). We follow the ASA's principles to present and interpret statistical significance [175].

Answering $\mathbf{RQ_{2.1}}$ 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 [150] (described in Section 2.1.1). 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.

To answer $\mathbf{RQ_{2.2}}$, we first group the activities according to the month of their occurrences.

[7]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.

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

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.

Answering $\mathbf{RQ_{2.1}}$ and $\mathbf{RQ_{2.2}}$ provides the data we need to answer $\mathbf{RQ_{2.3}}$. 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 $\mathbf{RQ_{2.2}}$ 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 $\times$ 36 months) data cases, in total. Each data case is in the following form:

$$< NewC_{im},\ BCT_{im},\ NewB_{im},\ BFR_{im},\ S\_com_{im},\ S\_org_{im},\ S\_sup_{im},\ S\_typ_{im} >$$

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

The $S - Com_{im}$ 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,

$$S\_com_{im} + S\_org_{im} + S\_sup_{im} + S\_typ_{im} = 1 \tag{2.1}$$

Answering $\mathbf{RQ_{2.3}}$ 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 [183]. Intuitively, each project has its own characteristics, so this study uses the project-specific fixed effects models.

$$NewC_{im} = \beta_1 \times S\_com_{im} + \beta_2 \times S\_org_{im} + \beta_3 \times S\_sup_{im} + \alpha_i + u_{it} \tag{2.2}$$

$$BCT_{im} = \beta_1 \times S\_com_{im} + \beta_2 \times S\_org_{im} + \beta_3 \times S\_sup_{im} + \alpha_i + u_{it} \tag{2.3}$$

$$NewB_{im} = \beta_1 \times S\_com_{im} + \beta_2 \times S\_org_{im} + \beta_3 \times S\_sup_{im} + \alpha_i + u_{it} \tag{2.4}$$

$$BFR_{im} = \beta_1 \times S\_com_{im} + \beta_2 \times S\_org_{im} + \beta_3 \times S\_sup_{im} + \alpha_i + u_{it} \tag{2.5}$$

Note that they do not include S_typ_{im} into Regression Equations (2.2)–(2.5). The reason is straightforward: the sum of S_typ_{im} and the other three is always "1" according to Eq. 2.1. Thus, it is perfectly correlated with the other three. Including it will lead to a significant multicollinearity problem.

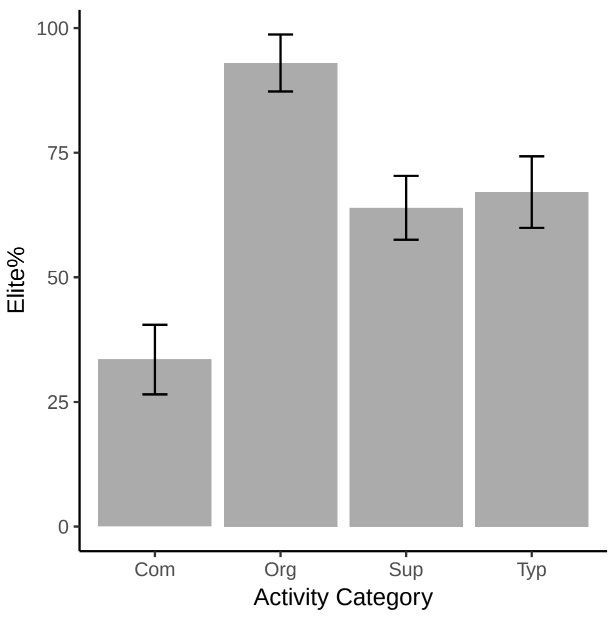


Figure 2.4: Elite developers' effort allocations in each category.

For each dependent variable, the least-squares dummy variables (LSDV) estimate the parameters in the project-specific fixed effects models. After finishing the model estimation, I 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 (`pFtest`), Hausman Test (`pHtest`), Heteroskedasticity testing, 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 for the two sub-samples. The results are reported accordingly. All the panel regressions, if not otherwise stated, are performed with `R`'s `plm` package [35].

2.1.2 Results and Findings

This section reports the results and findings on three **RQ**s respectively.

Elite Developers' Activities in Standalone Projects

Table 2.3: Descriptive statistics on activity amount for 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

Table 2.3 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., [49, 119, 170], and elite developers in the community contributed most of the source-code submission. In our sample, 67 percent of typical development tasks are performed by a project's elite developers.

In addition to elites' code submissions, 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 perform over 60% of supportive activities and even created 34% of communicative activities. See Figure 2.4 for the percentage distribution of elites' contributions, where error stands for standard errors of the mean (SEMs).

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 non-elite developer (see Fig. 2.5). There are 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 non-elite developer per month. Thus, on an individual basis, we argue that elite developers may have major impacts on projects based on their activity amount.

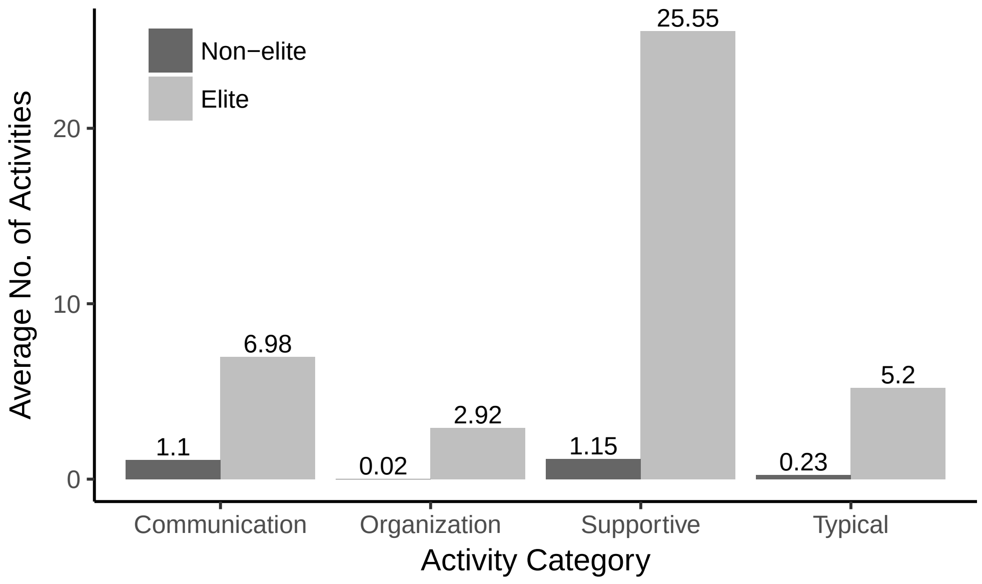


Figure 2.5: Comparison of an individual elite and non-elite developer's average activity amount during a project-month.

Based on the events in each category, we can answer $\mathbf{RQ_1}$ 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 documentation for the project, and also participating in discussions with teammates, external developers, and peripheral users.*

The Evolution of Elite Developers' Activities

Individual Activities of Elite: There are increasing trends in communicative, supportive, and organizational events for each elite developer group in our sample. Take the most complex project in the project sample, `Tensorflow`, as an example. The red lines in Figure 2.6 describe the changes in the 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 non-elite. For example, the blue lines in Figure 2.6 represent average activities' changing trends for `Tensorflow`'s non-elite 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 non-elite developers' typical technical activities is similar (stable after the initial bursts). However, their initial bursts happened after the elites. This study excludes the changes in non-elite developers' organizational activities since the non-elite do not have the privileges to perform nearly any of the activities in this category.

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

Comparing growth rates of the four types of activities: As mentioned in Section 2.1.1, the average monthly growth rates of activities per elite developer can be calculated

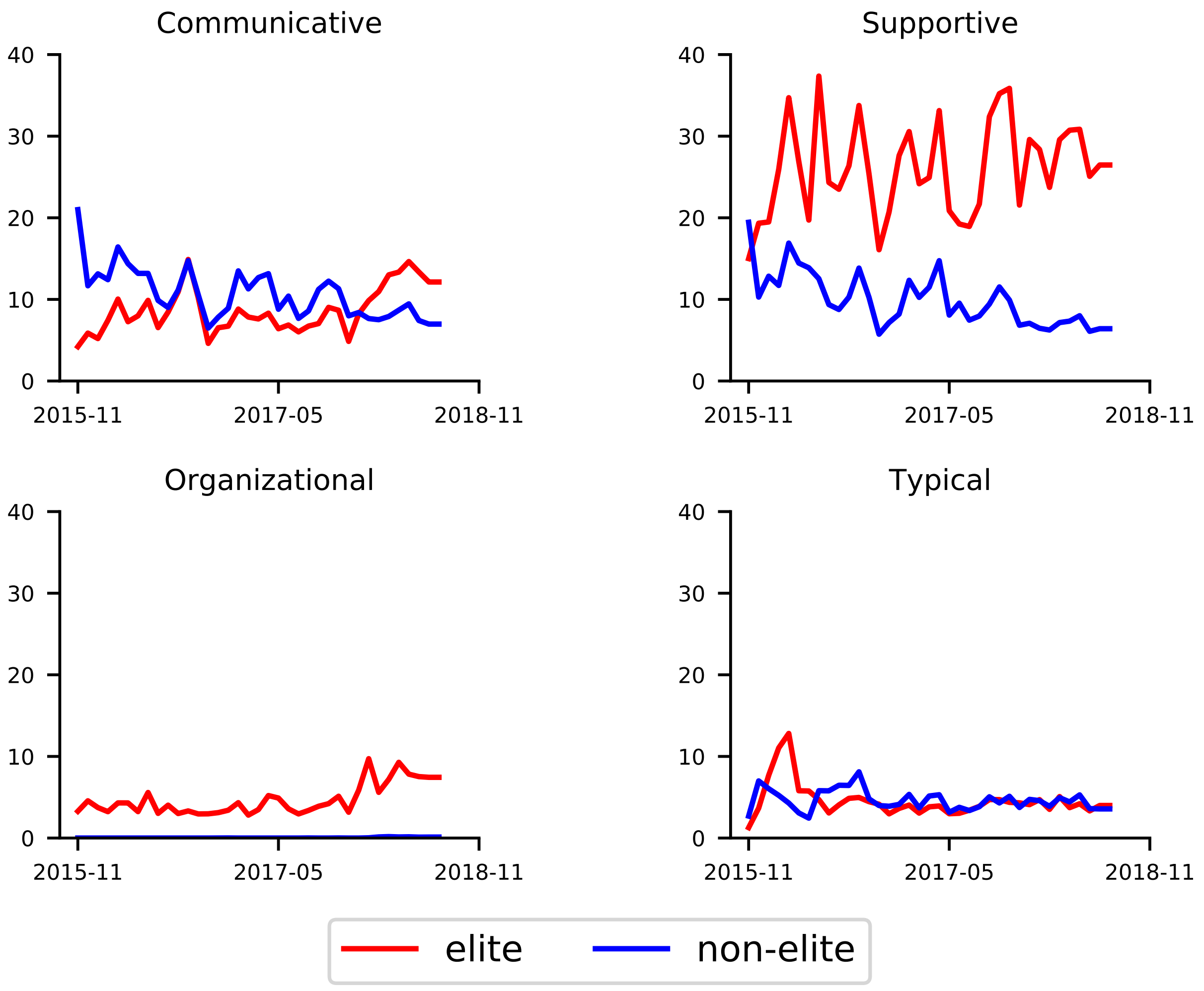


Figure 2.6: Trends of individual developer's activities in the four activity categories of `Tensorflow`.

over the communicative, supportive, and typical activities[8] for each project. Thus, we have 20 growth rates for these three categories of activities. We then perform the one-way ANOVA to test if there is any difference in growth rates.

The results shows significant differences ($F_{(2,57)} = 8.452$, $p < 0.001$). The post hoc analyses were performed by Tukey's HSD test to identify the differences between the three categories. The results indicate 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. Sup-

[8]For organizational activities, many months do not record such a type of activities, especially in less popular projects. This prohibits us from calculating the growth rate.

portive: $p = 0.002$). Moreover, 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 performs half of the technical work they used to 36 months ago. Meanwhile, their communicative and supportive work doubled in the same period.

This study does not perform the same ANOVA procedures on the non-elite data for cross-group comparisons (i.e., elite vs. non-elite) due to practical constraints. In many months, the non-elite 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 in the three types of non-technical activities over time, while the numbers of non-elites technical activities in each project-month do increase over time.

Based on the result of the one-way ANOVA test and Tukey's HSD test, we can answer $\mathbf{RQ_{2.2}}$ as follows:

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

Elite Developers' Activity Impact on Project Outcomes

Findings of $\mathbf{RQ_{2.3}}$ can be presented with a series of regression models (Table 2.4 and 2.5) 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 2.1.1. This study also performs regression diagnostics to empirically examine the justification of using fixed effects models in model development. For all 12 models, fixed effects models are better choices than pooled OLS and random models.

Table 2.4: Regression models for project productivity.

	Whole Sample		Sub-sample (Non-Company)		Sub-sample (Company)	
	New Commit Model P1 (β) (*SE*)	Bug Cycle Time Model P2 (β) (*SE*)	New Commit Model P3 (β) (*SE*)	Bug Cycle Time Model P4 (β) (*SE*)	New Commit Model P5 (β) (*SE*)	Bug Cycle Time Model P6 (β) (*SE*)
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)
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)
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 paper concise, we do not include them, instead, we show the significant levels of them.
†: For Model 1 – 2: degrees of freedom (*DF*s) are (23, 697); for Model 3 – 4: degrees of freedom are (12, 312); for Model 5 – 6: degrees of freedom are (14, 382).

Regression results of project productivity: Table 2.4 summarizes the results of the regression models for the two project productivity indicators: the number of new commits of project i in 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, thus representing 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–P6 represent sub-sample regression results. Below we describe what these models indicate.

Project productivity—whole sample regression results: In Model P1, two independent variables (S_com_{im}, S_sup_{im}) are significant; and both have negative regression coefficients (-155.96, -138.21). This implies **negative correlations** between the effort of elite developers in communicative and supportive activities and the number of new commits in each project-month. A possible interpretation of the results is that when elite developers invest more effort into non-technical activities, such as communicative and supportive ones, they might leave less time to continue their contribution to the code base; thus, the project may have fewer new commits, i.e., productivity loss.

In Model P2, for which the dependent variable is BCT_{im}, two independent variables (S_org_{im}, S_sup_{im}) are significant. S_org_{im} has a negative regression coefficient (-148.61), while S_sup_{im} has a positive coefficient (473.60). First, 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. Second, given that the activities of managing bug fixes and code reviews are in the "organizational" category (see Figure 2.2), more elite efforts in this category might help to shorten the bug cycle time. Third, 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 S_sup_{im}'s effect is much stronger than S_org_{im}'s and its shares are often much more than S_org_{im}'s (Avg.: 0.61 vs. 0.03), we could expect an overall effect of longer bug cycle time, i.e., productivity loss.

Project productivity—sub-sample regression results: The regression results in Models P3-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-sponsor projects' data (Models P3 and P4). In Model P3, 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-sponsor projects' data (Models P5 and P6), correlations in Model P5 are as same as those in Model P1. However, in Model P6, 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-sponsor projects.

Regression results of project quality: Table 2.5 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-Q6 represent sub-sample regression results.

Project quality—whole sample regression results: In Model Q1, the quality indicator is $NewC_{im}$. There are two significant independent variables (S_org_{im}, 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 non-technical work for supporting other community developers and users may make the elite spend less time working on code. Meanwhile, the project may receive a large number of suggested changes and patches from non-elite developers, but their code may contain more bugs, i.e., quality loss.

In Model Q2, the quality indicator is BFR_{im}. Two independent variables are significant (S_com_{im}, S_sup_{im}). 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, S_sup_{im}'s coefficient is positive, indicating *positive correlations* between the elite's efforts in supportive activities and each month's bug fix rate. Interpreting such correlations may be a bit tricky. For the negative correlations between S_com_{im} and BFR_{im}, we can interpret them in a way similar to the previous ones. The positive correlations between S_sup_{im} and BFR_{im} may suggest that: by putting more efforts into supportive activities, elite developers help to make the defect removal process more effective.

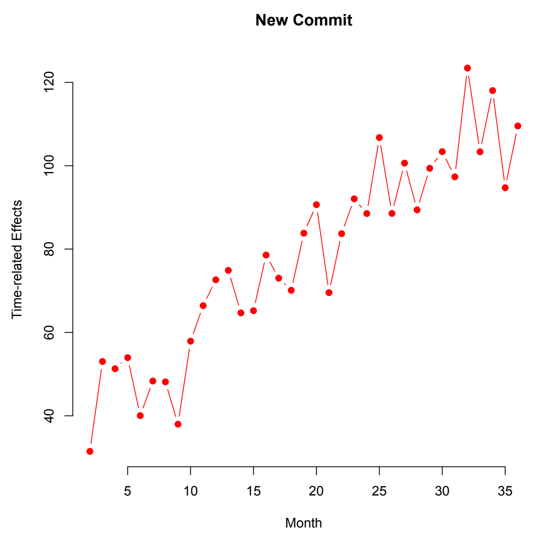
Since S_com_{im}'s share are only about a quarter of S_sup_{im}'s (Avg.: 0.16 vs. 0.61) and its negative coefficient is just twice of S_sup_{im}'s (-2.24 vs. 1.12), we could expect an overall on average effect of higher bug fix rate (quality gain).

Project quality—sub-sample regression results: Again, the whole-sample data set was split into two sub-sample data sets according to whether a project is sponsored by a commercial company, and then developed regression models respectively. Models Q3 and

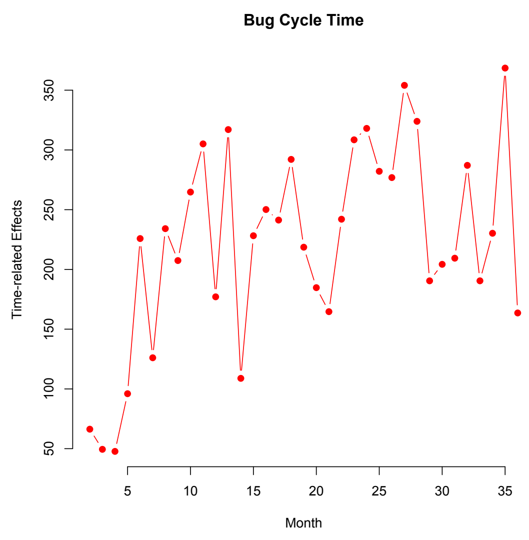
Table 2.5: Regression models for project quality.

	Whole Sample		Sub-sample (Non-Company)		Sub-sample (Company)	
	New Bug	Bug Fix Rate	New Bug	Bug Fix Rate	New Bug	Bug Fix Rate
	Model Q1 (β)	Model Q2 (β)	Model Q3 (β)	Model Q4 (β)	Model Q5 (β)	Model Q6 (β)
	(*SE*)	(*SE*)	(*SE*)	(*SE*)	(*SE*)	(*SE*)
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)
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)
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***

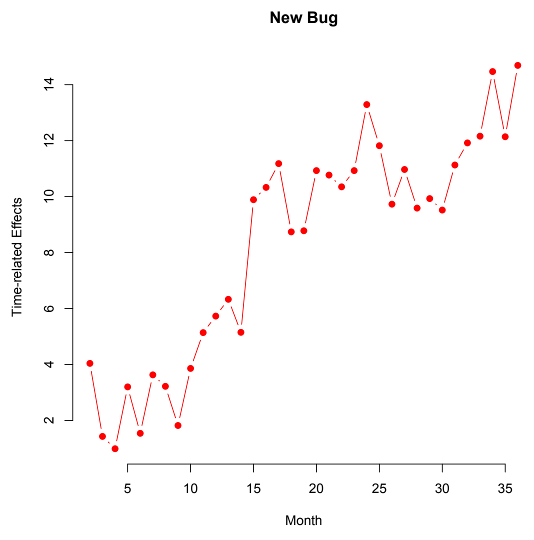
*: $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$.
¶: The unobserved time-invariant effects are a vector rather than a single coefficient. To keep the paper concise, we do not include them, instead, we show their significant levels.
†: For Model 1 – 2: degrees of freedom (DFs) are (23, 697); for Model 3 – 4: degrees of freedom are (12, 312); for Model 5 – 6: degrees of freedom are (14, 382).



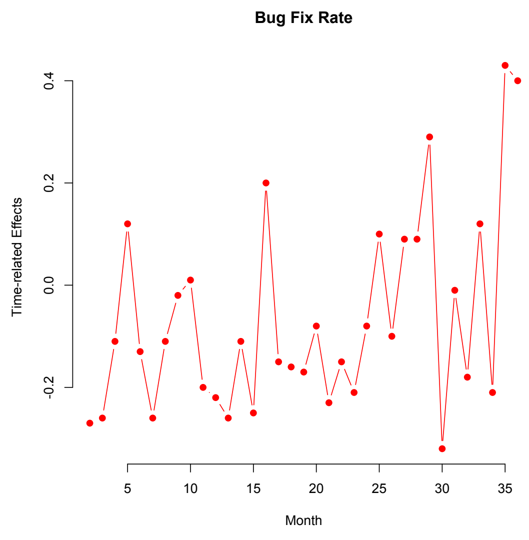
(a)



(b)



(c)



(d)

Figure 2.7: Changes of time-related effects in the regression models.

Q4 are based on non-company-sponsored projects' data. In Model Q3, only 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.

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 2.4; and Models Q1, Q2 in Table 2.5).

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 (Fig. 2.7(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 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 do not have significant patterns (Figure 2.7(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 (Fig. 2.7(a)). The time-related effects are positive and increasing in general (Fig. 2.7(c)), indicating the impact of elite developers' activities on the number of the 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 (Fig. 2.7(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.

Based on the above results, we can answer $\mathbf{RQ_{2.3}}$ 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. Combining 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 non-elite 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.

2.2 Elite Developers in OSS Ecosystems

Though the first empirical study investigates how elite developers act and impact several standalone OSS projects, more OSS projects succeed as a part of multiple interrelated projects in an ecosystem [60]. An OSS ecosystem refers to a collection of OSS projects that are designed, developed, and evolved in a common platform [22, 106]. For example, the Hadoop ecosystem consists of over 100 projects [95]. The ecosystem model blurs the traditional boundaries between development entities by providing a framework that enables different actors, e.g., commercial stakeholders, voluntary contributors, and end-users, to work interdependently [72, 78, 109]. We have witnessed many successes of OSS ecosystems, to name a few, Android, Hadoop, Node.js, R, etc., which have received much attention from academia and industry; research into it started flourishing since the mid-2000s [109].

An OSS ecosystem is indeed a socio-technical system where humans, technologies, and artifacts interact consistently to produce complex social dynamics [73, 140]. Social ecology theorists often argue that while members' activities shape the many aspects (e.g., cultures and outcomes) of the ecosystem they belong to, the ecosystem also influences and (re)constructs its members' behaviors [54, 153, 154]. For instance, how they allocate their efforts between technical and non-technical activities [44]. Such ecosystem-specific activity patterns would in turn influence projects' outcomes.

However, individual-level activities within and across OSS ecosystems have not yet been well investigated. Even for a few studies considering developers' perceptions and behaviors in an ecosystem [134], their inquiries are still centralized on the perceptions and behaviors triggered by changes of artifacts (e.g., source code) but neglect various activities beyond technical contributions. Understanding individual-level activities in a full spectrum of contributions and their relationships with project outcomes have particular importance in open source development [9], particularly at the ecosystem level. Such understanding would inform the design of effective and efficient work practices, management mechanisms, and tools to improve the overall performance of OSS projects and the ecosystems they belong to [9, 61, 145].

Among all contributors, some privileged developers have been consistently recognized as the most important project personnel in SE literature due to their significant contributions [37, 39, 119]. The study in the prior section conceptualizes a group as "*elite developers*" who hold project management privileges and extensive access to the artifact repository and investigate the relationships between their activities and project outcomes. Further, here I extend its standalone project angle by considering the practice within OSS ecosystems. Using developers' ecosystem-wide activity data, the current study seeks to develop an empirical understanding of their various contributing activities and the impacts of their effort allocations over these activities on technical outcomes, with a particular emphasis on cross-ecosystem similarities and differences. Thus, we ask the following two research questions:

RQ$_{\mathbf{2.4}}$ *Are there any differences regarding the elite developers' activity profiles across the five targeted OSS ecosystems? If so, what are they?*

RQ$_{\mathbf{2.5}}$ *What are the relationships between elite developers' effort allocations on different activities and project outcomes in the five target OSS ecosystems (productivity and quality)? Are there any differences regarding such relationships?*

To answer the above research questions, this study selected five open source ecosystems (Amazon AWS, Eclipse, npm/Node.js, Firefox Add-ons, and Python Data Science). Each ecosystem has a unique position on the spectrum of software systems. This study samples a number of *software development* projects in each ecosystem and then compiles a data set of 65 projects hosted on GitHub that contains all detailed atomic activity data from January 2016 to December 2018. Based on prior studies [150, 172], we classify these fine-grained raw activities into four categories: *communicative*, *organizational*, *supportive*, and *typical.*

This study empirically characterizes and compares elite developers' activity profiles across sampled ecosystems. By re-building a series of econometric models, I further explore the relationships between elite developers' effort allocations and project outcomes in these five ecosystems. The study results describe such relationships and reveal that these relationships vary across ecosystems in terms of effect and strength.

2.2.1 Empirical Study Design on Major OSS Ecosystems

This study extends the prior study and also reuses its methodology including the event-category mapping, and productivity and quality proxies. This section briefly introduces the entire research method and explains the target ecosystem selection and project sampling process.

Target Ecosystems and Projects

This study selects five major OSS ecosystems that have been widely studied in the software engineering literature as the targets of the study [7, 107, 127, 160]. They are Amazon AWS, Eclipse, npm/Node.js, Firefox Add-ons, and Python Data Science. These ecosystems and projects were selected for the following two considerations.

First, this study targets ecosystems representing a wide range of domains of software systems because prior literature suggests that investigating software development practices across domains should improve our understanding of the board spectrum of OSS development [167]. Thus, the selected five ecosystems intend to satisfy this requirement: Amazon AWS represents systems for cloud computing infrastructures. Eclipse is a collection of productivity tools for software developers. Firefox Add-ons represent the increasingly popular trend of customizing browsers. npm/Node.js aims to build scalable, dynamic network applications. It

Table 2.6: List of sampled OSS ecosystems and projects.

Ecosystem	**Description**	**No.**	**Projects**
Amazon AWS	A framework provides cloud computing abstraction, infrastructure, and tools.	17	*amazon-ecs-agent*, *amazon-ecs-cli*, *aws-cli*, *aws-fpga*, *aws-lambda-dotnet*, *aws-parallelcluster*, *aws-sdk-cpp*, *aws-sdk-go*, *aws-sdk-java*, *aws-sdk-java-v2*, *aws-sdk-js*, *aws-sdk-net*, *aws-sdk-php*, *aws-sdk-ruby*, *chalice*, *sagemaker-python-sdk*
Eclipse	An IDE supporting multiple programming languages and plugins.	14	*californium*, *che*, *buildship*, *eclipse-collections*, *eclipse.jdt.ls*, *golo-lang*, *jetty.project*, *mosquitto*, *omr*, *paho.mqtt.c*, *paho.mqtt.golang*, *paho.mqtt.java*, *paho.mqtt.javascript*, *paho.mqtt.python*
Firefox Add-ons	A collection of Firefox add-ons and components.	8	*activity-stream*, *addons-frontend*, *addons-linter*, *fxa*, *geckodriver*, *multi-account-containers*, *normandy*, *python_mozetl*
npm/Node	A server-side JavaScript framework and its package manager.	12	*docker-node*, *http-parser*, *llnode*, *marky-markdown*, *nan*, *node*, *node-addon-api*, *node-chakracore*, *node-gyp*, *node-semver*, *node-tar*, *readable-stream*
Python DataSci	A collection of data science projects for Python.	14	*bokeh*, *glow*, *keras*, *matplotlib*, *nltk*, *numpy*, *pandas*, *pytorch*, *scikit-learn*, *scipy*, *statsmodels*, *text*, *sympy*, *vision*, *xgboost*

popularizes the server-side JavaScript environment and brings a significant web development paradigm shift in the last ten years. Firefox Add-ons and npm/Node.js projects are usually lightweight and deployed in the form of source code. Python Data Science is a collection of mainstream productivity tools for data scientists. Second, these ecosystems should have a substantial amount of active software development and OSS projects hosting on GitHub. In addition to enabling us to collect data conveniently, it helps to improve the study's internal validity of the study. Since all projects are using the same terminology when referring to their members' activities, the potential misinterpretations and mismatches of the activities and events could be minimized.

This study samples a number of projects from each ecosystem through a *purposive sampling process* [53]. These projects are selected based on project status, contribution model, and data availability. First, selected projects must be active projects using the pull request model to solicit contributions. Second, there must be a substantial amount of project activities from 2016 to 2018 for each sampled project. Therefore, we require all selected projects to have at least *100* issues, *50* pull requests and *100* commits to ensuring that the project is active during the sampled period. Third, in order to avoid misleading event data, we only include *software development* repositories and exclude ones identified as *experimental*, *web*, *academic* or *storage* (classification by Kalliamvakou et al. [83]). Fourth, to ensure data consistency, we exclude all projects using non-GitHub issue trackers. To verify the above requirements, we manually examine every selected repository. Such criteria guarantee that we can retrieve a sufficient amount of elite members' activities for analysis in each repository. Finally, there are 65 projects in the sample after applying all the above criteria from the five chosen ecosystems (See Table 2.6).

Data Collection and Preparation

Figure 2.8 presents the entire process of data collection and preparation. The process consists of two major steps: first, collecting data from online databases and repositories; second, coding and modeling the data to compile the data sets for answering the two research questions of this section.

This study's data collection methods and its data cleaning and preparation procedure are similar to the previous study on standalone projects. Therefore, please refer to the prior Section 2.1.1 for details of data collection, event mapping, and pre-processing.

Data Analysis

Table 2.7 presents the mappings between the **RQs** of this study and corresponding analysis methods. This section introduces them in detail.

Table 2.7: Research questions and corresponding data analysis methods for ecosystem study.

RQs	Data Analysis Methods
$\mathbf{RQ_{2.4}}$	Descriptive statistics, one-way ANOVA
$\mathbf{RQ_{2.5}}$	Panel Regressions (Fixed effects or random effects to be determined empirically)

Analysis methods for $\mathbf{RQ_1}$. Answering $\mathbf{RQ_{2.4}}$ does not require complicated analysis,

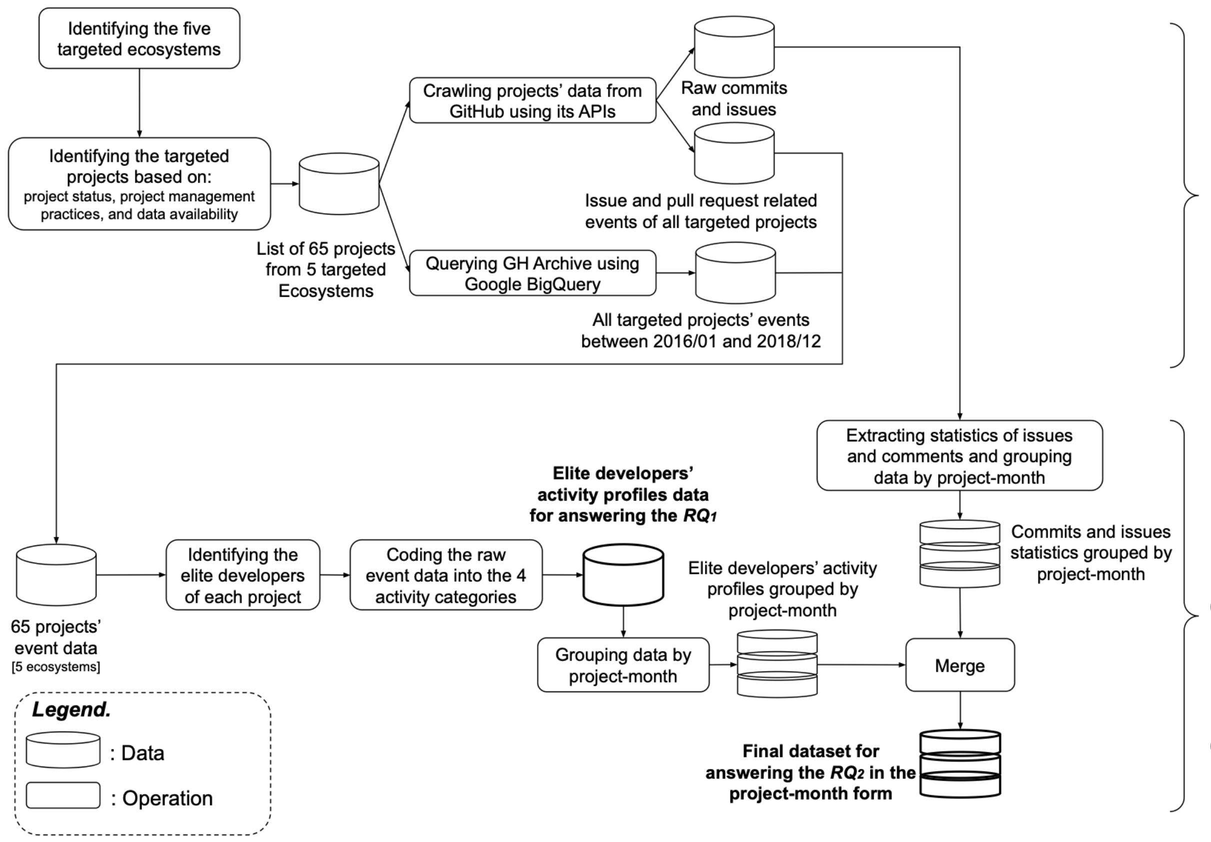

Identifying the five targeted ecosystems
Identifying the targeted projects based on: project status, project management practices, and data availability
List of 65 projects from 5 targeted Ecosystems
Crawling projects' data from GitHub using its APIs
Raw commits and issues
Issue and pull request related events of all targeted projects
Querying GH Archive using Google BigQuery
All targeted projects' events between 2016/01 and 2018/12
Data Collection
Extracting statistics of issues and comments and grouping data by project-month
Commits and issues statistics grouped by project-month
Elite developers' activity profiles data for answering the RQ1
65 projects' event data [5 ecosystems]
Identifying the elite developers of each project
Coding the raw event data into the 4 activity categories
Elite developers' activity profiles grouped by project-month
Grouping data by project-month
Merge
Final dataset for answering the RQ2 in the project-month form
Data Coding and Modeling
Legend.
: Data
: Operation

Figure 2.8: Data collection and preparation process.

and is similar to $\mathbf{RQ_{2.1}}$ and $\mathbf{RQ_{2.2}}$. First, this study uses descriptive statistics to describe elite developers' activities over the four activity categories according to their ecosystems. I also append information about the number of elite members every month in the scale of an OSS ecosystem. Then we calculate the average number of activities performed by an elite member in a project-month and its standard deviations on each category for every ecosystem. For each ecosystem, this study presents also the total activities (performed by both elite and non-elite) in each category and plots the proportions of elite developers' activities. Finally, one-way ANOVA was conducted to test whether there were significant differences between ecosystems regarding elite developers' proportion of activities. When a test's result was significant, post hoc analysis was performed to investigate what contributed to the differences.

Analysis Methods for $\mathbf{RQ_{2.5}}$. Answering $RQ_{2.5}$ requires identifying relationships between elite developers' effort allocations and project outcomes. Our dataset of 6-tuples includes measures of project outcomes and effort allocations. One of the most suitable choices is building regression models where project outcomes are dependent variables, and effort allocations are independent variables. However, note that the data set is unbalanced panel data (cross-sectional: from 65 projects belonging to 5 ecosystems; longitudinal: 13-36 months). Thus, to correctly identify the relationships, this study also uses panel regressions to model the data [183]. As one of the advanced econometrics techniques, panel regression models and isolates the effects from confounding factors implicitly and inherently, for instance, project-specific factors (domain, etc.) are treated as unobserved time-invariant in regressions. Thus, the three independent variables' effects could be precisely estimated without worrying about potential interactions and selective biases with these confounding factors [183].

Given that this study also intends to compare the differences across ecosystems, it builds models for each ecosystem separately. Note that the sum of the four independent variables is always "1" in each data case, therefore, this study only uses the first three of them (including

P_com_{im}, P_org_{im}, and P_sup_{im}) to avoid multicollinearity in regression models. Moreover, to avoid potential model misspecification, for each data sample, this study builds both fixed and random effects models when estimating the correlations between elite developers' activity profiles and project outcomes and then evaluates them empirically through model diagnostics [138]. Therefore, this study built 24 models for 12 regression tasks (2 project outcomes $\times$ 6 data samples [5 ecosystems + all ecosystems] = 12 regression tasks). Then, after developing all models, we performed model diagnostics (Hausman specification test) to empirically examine which type of models are better for each specific regression task. Section 2.2.2 only reports the better-fitting model for each task only (see Table 2.9).

2.2.2 Results and Findings

Elite Developers' Activity Profiles in Five Ecosystems

Table 2.8 presents descriptive statistics for elite developers' activities in the five ecosystems. Column 4, **Avg.$_{\textbf{Person-Month}}$**, lists the average number of actions an elite developer performs in a month with standard deviations in Column 4. Among all ecosystems, the elite from the Python Data Science ecosystem work a little bit "harder" than those from other ecosystems: they perform more activities on average, particularly on the communicative (24.54) and supportive (55.23) tasks. The elite from the Firefox Add-ons ecosystem tops on organizational activities (4.83) while their peers from the Amazon AWS ecosystem are the champions in typical activities (8.47).

Figure 2.9 displays the proportions that the elite's activities account for (of all activities) in the five ecosystems grouped by the activity types. This figure does not include the proportions of organizational activities since most organizational activities required permissions that non-elites do not have. The elites from the Firefox Add-ons ecosystem had the highest proportions across all categories. The non-elites from this ecosystem are not quite active.

Table 2.8: Descriptive statistics for elite developers' activities.

Ecosystem	**Activity**	**N**	**Avg.**$_{\textbf{Person-Month}}$	**Std.**(σ)
Amazon AWS	*Com*	19407	6.89	7.13
	Org	7164	1.85	2.71
	Sup	50213	17.86	17.72
	Typ	19561	8.47	11.49
Eclipse	*Com*	46007	9.01	8.98
	Org	23620	3.10	5.21
	Sup	104428	20.91	24.55
	Typ	21406	7.54	10.58
Firefox Add-ons	*Com*	32478	9.41	7.56
	Org	20100	4.83	6.12
	Sup	101727	26.97	25.98
	Typ	22318	7.85	6.84
npm/Node.js	*Com*	157099	10.88	19.18
	Org	6796	0.78	1.60
	Sup	128804	12.28	17.88
	Typ	15421	3.45	7.85
Python Data Science	*Com*	160031	24.54	28.29
	Org	24434	0.83	1.80
	Sup	289178	55.23	91.61
	Typ	53041	0.93	14.64

Since Figure 2.9 suggests there are some visible differences across different ecosystems, this study further explores these differences using the one-way between-subjects ANOVA test.

Three ANOVA tests were performed for each type of activity. For each ecosystem, a list of numbers represents the proportions of elite developers' activities of all projects in this ecosystem, one per project. Each test was based on the comparison of five lists of proportions. Thus, an ANOVA test compares the effect of the ecosystem (*Independent Variable*) on the elite's proportion of activity for each ecosystem on one of the three types of activity (*Dependent Variable*). All three ANOVA tests show significant effects of IV on DV at the significant level of 0.05: (1) communicative activities, $F_{4,60} = 5.22$, $p < 0.01$; (2) supportive activities, $F_{4,60} = 3.21$, $p < 0.05$; (3) typical activities, $F_{4,60} = 5.22$, $p < 0.001$.

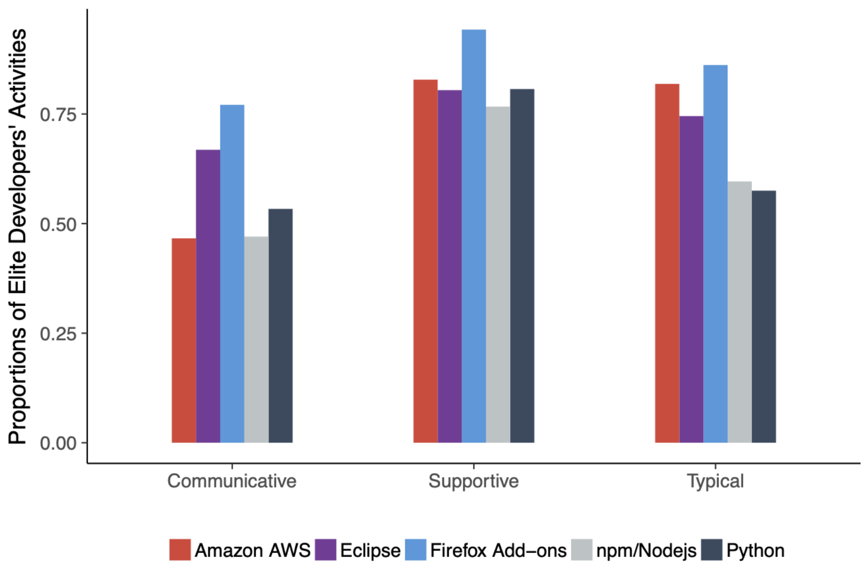


Figure 2.9: The proportions of elite developers' activities.

Then post hoc comparisons were performed by using the Tukey HSD test to examine which pairs contribute to the differences across ecosystems identified in the above ANOVA tests. For communicative activities, the results suggested four pairs with significant differences, which were: (Firefox Add-ons vs. Amazon AWS: $\Delta = 0.30,\ p < 0.001$), (Firefox Add-ons vs. Eclipse: $\Delta = 0.24,\ p < 0.05$) (Firefox Add-ons vs. npm/Node.js, $\Delta = 0.30,\ p < 0.01$) and (Firefox Add-ons vs. Python Data Science: $\Delta = 0.24,\ p < 0.05$). For supportive activities, there is one pair with significant differences, which is (Firefox Add-ons vs. npm/Node.js: $\Delta = 0.18,\ p < 0.01$). For typical activities, there are four pairs with significant differences, which are (Amazon AWS vs. npm/Node.js: $\Delta = 0.22,\ p < 0.01$), (Amazon AWS vs. Python Data Science, $\Delta = 0.24,\ p < 0.001$), (Firefox Add-ons vs. npm/Node.js, $\Delta = 0.27,\ p < 0.01$), and (Firefox Add-ons vs. Python Data Science, $\Delta = 0.29,\ p < 0.001$).

Answers to $\mathbf{RQ_{2.4}}$. Based on the above discussions, $\mathbf{RQ_{2.4}}$ can be answered as follows:

For all ecosystems, elite developers consistently perform substantial proportions of activities in all three activity categories. However, there are significant differences across ecosystems in each activity category. Particularly, the elites' performance from the *Firefox Add-ons* ecosystem accounts for higher proportions of activities than the elites from other ecosystems on all types of activities, and *Amazon AWS*'s elite developers' proportion on typical activities are significantly higher than *npm/Node.js*' and *Python Data Science*'s elite developers.

The Relationships between Effort Allocations and Project Outcomes

This section first summarizes multiple panel regression models assessing the relationships between elite developers' effort allocations and project outcomes. Then, it also highlights the similarities and differences in the relationships across the sampled ecosystems.

Results by ecosystem. Table 2.9.a–e summarize panel regression models. Each of these sub-tables corresponds to a targeted ecosystem. Presented models are selected by the Hausman specification test, and the letters "F" (Fixed Effects) and "R" (Random Effects) in the parentheses next to model names indicate the type of models. The panel regression results of the whole sample that contains data cases from all ecosystems attach at the end in Table 2.9.f. For each model, result tables also append the coefficients (including *std err.*) of independent variables, as well as model attributes *Adj.* R^2, F with df. Last but not least, these results employ Cohen's f^2 as the indicator of the effect size: 0.02, 0.15, and 0.35 as the thresholds of *small*, *medium*, and *strong* effects according to conventional effect levels [32].

For projects in the ecosystem of *Amazon AWS*, their elite developers' efforts in communicative ($\beta = -120.56$, $p < 0.001$) and supportive ($\beta = -91.71$, $p < 0.001$) activities significantly predict the project productivity in terms of the number of new commits every month. Its regression model (a.Model 1) explains that there is 18% of variance (*Adj.* $R^2 = 0.18$,

Table 2.9: Panel regression results.

(a) Amazon AWS
(Pro. = 17, Obs. = 515, T = 13–36)

	Model 1 (R) Commit (C_{im})	Model 2 (R) Bug (B_{im})
Intercept	117.29***	4.36***
	(9.09)	(1.09)
S_com_{im}	−120.56***	−1.82
	(14.17)	(1.85)
S_org_{im}	−29.13	6.06*
	(18.14)	(2.37)
S_sup_{im}	−91.71***	0.78
	(10.15)	(1.32)
Adjusted R^2	0.18	0.02
Cohen's f^2	0.22	0.02
F	114.74***	8.98*
df	(3)	(3)

(b) Eclipse
(Pro. = 14, Obs. = 460, T = 23–36)

	Model 1 (F) Commit (C_{im})	Model 2 (R) Bug (B_{im})
Intercept	−.−	2.57
	(−.−)	(9.36)
S_com_{im}	−92.41***	−10.64
	(18.37)	(8.03)
S_org_{im}	−74.12	116.67***
	(38.30)	(16.71)
S_sup_{im}	−79.56***	14.12
	(17.77)	(7.77)
Adjusted R^2	0.03	0.13
Cohen's f^2	0.03	0.15
F	10.33***	64.72***
df	(3, 443)	(3)

(c) npm/Node.js
(Pro. = 12, Obs. = 359, T = 21–36)

	Model 1 (R) Commit (C_{im})	Model 2 (R) Bug (B_{im})
Intercept	140.37**	6.45
	(47.73)	(8.09)
S_com_{im}	−102.60**	0.52
	(27.63)	(3.89)
S_org_{im}	117.08	8.82
	(66.96)	(9.43)
S_sup_{im}	−83.80***	2.41
	(31.20)	(4.39)
Adjusted R^2	0.06	−0.01
Cohen's f^2	0.06	0.00
F	26.66***	1.15
df	(3)	(3)

(d) Firefox Add-ons
(Pro. = 8, Obs. = 260, T = 21–36)

	Model 1 (F) Commit (C_{im})	Model 2 (R) Bug (B_{im})
Intercept	−.−	−5.91
	(−.−)	(7.63)
S_com_{im}	−40.45	2.57
	(71.63)	(12.49)
S_org_{im}	143.31	24.92
	(80.35)	(14.64)
S_sup_{im}	−75.04	25.81**
	(54.72)	(9.11)
Adjusted R^2	0.00	0.03
Cohen's f^2	0.00	0.03
F	2.94	10.98*
df	(3, 249)	(3)

(e) Python Data Sci.
(Pro. = 14, Obs. = 474, T = 24–36)

	Model 1 (R) Commit (C_{im})	Model 2 (F) Bug (B_{im})
Intercept	416.817***	−.−
	(58.70)	(−.−)
S_com_{im}	−288.35***	23.91*
	(66.46)	(10.77)
S_org_{im}	−13.32	208.71***
	(126.10)	(20.40)
S_sup_{im}	−328.21***	8.17
	(59.94)	(9.71)
Adjusted R^2	0.06	0.18
Cohen's f^2	0.06	0.22
F	33.07***	39.32***
df	(3)	(3, 457)

(f) All Ecosystems
(Pro. = 65, Obs. = 2068, T = 13–36)

	Model 1 (R) Commit (C_{im})	Model 2 (F) Bug (B_{im})
Intercept	166.63***	−.−
	(16.41)	(−.−)
S_com_{im}	−115.10***	1.04
	(13.99)	(3.24)
S_org_{im}	10.09	56.35***
	(25.54)	(5.90)
S_sup_{im}	−113.19***	8.37**
	(13.19)	(3.05)
Adjusted R^2	0.04	0.02
Cohen's f^2	0.04	0.02
F	95.96***	33.47***
df	(3)	(3, 2000)

(F): Fixed effects model, (R): Random effects model; $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$

$F_3 = 114.74$, $p < 0.001$) with a medium effect (Cohen's $f^2 = 0.22$). The regression coefficients are both negative, which indicates that their project's productivity has a negative association with more effort into communicative and supportive activities by their elite developers. On the other hand, there are few significant predictive relationships between a project's quality and effort developers' effort allocation. Only the efforts allocated to organizational activities are significant ($\beta = 6.06$, $p < 0.05$), and the model (a.Model 2) only explains 2% of variances (*Adj.* $R^2 = 0.02$, $F_3 = 8.98$, $p < 0.05$) with a small (almost not significant) effect (Cohen's $f^2 = 0.02$). Hence, Amazon AWS' project quality cannot be significantly predicted by elite developers' effort allocations.

For the second ecosystem—*Eclipse*, its elite developers' efforts in communicative ($\beta = -92.41$, $p < 0.001$) and supportive ($\beta = -79.56$, $p < 0.001$) activities can significantly predict the project productivity. However, the effect size is at a small scale. The regression model (b.Model 1) explains 3% of variance (*Adj.* $R^2 = 0.03$, $F_{3,443} = 10.33$, $p < 0.001$), with a small effect (Cohen's $f^2 = 0.03$). Hence, *Eclipse*'s project productivity has a negative association with more efforts into communicative and supportive activities by their elite members with a small effect size. Moreover, for their projects' quality, the efforts in organizational activities are significant ($\beta = 6.06$, $p < 0.05$), and the model (b.Model 2) explains a substantial amount of variances (*Adj.* $R^2 = 0.13$, $F_3 = 64.72$, $p < 0.001$). The results indicate that *Eclipse* projects' quality has a negative association with their elite developers' increasing effort into organizational activities. There is no significant predictive relationship found with other types of activities.

In the third ecosystem—*npm/Node.js*, elite developers' effort allocations show similar effects on project productivity as the previous two. The increasing efforts in communicative ($\beta = -102.60$, $p < 0.01$) and supportive ($\beta = -83.80$, $p < 0.001$) activities has a negative correlation with project productivity. The regression model (c.Model 1) explains 6% of variance (*Adj.* $R^2 = 0.06$, $F_3 = 26.66$, $p < 0.001$), with a small-level effect size (Cohen's

$f^2 = 0.06$). Nevertheless, in this ecosystem, there is no activity type that could significantly predict its project quality. The project quality model even records a negative *adjusted* R^2. Therefore, there is no significant predictive relationship between the allocations of elite developers' activities and *npm/Node.js'* project quality.

For the fourth ecosystem—*Firefox Add-ons*, its elite developers' effort allocations do not have a significant predictive relationship with the project productivity. None of its *Independent Variables* shows significance, and the model's *adjusted* R^2 is 0 (d.Model 1). For project quality, efforts in supportive activities show a significant correlation ($\beta = 25.81$, $p < 0.01$), but its model (d.Model 2) only can explain 3% of variance (*Adj.* $R^2 = 0.06$, $F_3 = 26.66$, $p < 0.001$) with a small level effect (Cohen's $f^2 = 0.03$). Therefore, although efforts of supportive activities from elite developers in *Firefox Add-ons* have negative associations with their project quality, its effect size is at a small level.

Activity and effort trends of the last ecosystem—*Python Data Science*'s elite developers significantly predicted both their project productivity and quality. For productivity, it is similar to the other previous ecosystems: efforts in communicative ($\beta = -288.358$, $p < 0.001$) and supportive ($\beta = -328.21$, $p < 0.001$) activities significantly predicted project productivity. Its model (e.Model 1) explains 6% of variance (*Adj.* $R^2 = 0.06$, $F_3 = 33.07$, $p < 0.001$) with a small level effect size (Cohen's $f^2 = 0.06$). Given the negative coefficients, the result suggests that the *Python Data Science* project's productivity is negatively correlated with elite developers' efforts in communicative and supportive activities. In addition, their efforts in communicative ($\beta = 23.91$, $p < 0.05$) and organizational ($\beta = 208.71$, $p < 0.001$) activities significantly predict their project quality. The model (e.Model 2) can explain 18% of variance (*Adj.* $R^2 = 0.18$, $F_{3,457} = 39.32$, $p < 0.001$) with medium effects (Cohen's $f^2 = 0.22$). Since these models' coefficients are both positive, this ecosystem's project quality has negative associations with elite developers' efforts in communicative and organizational activities in *Python Data Science.*

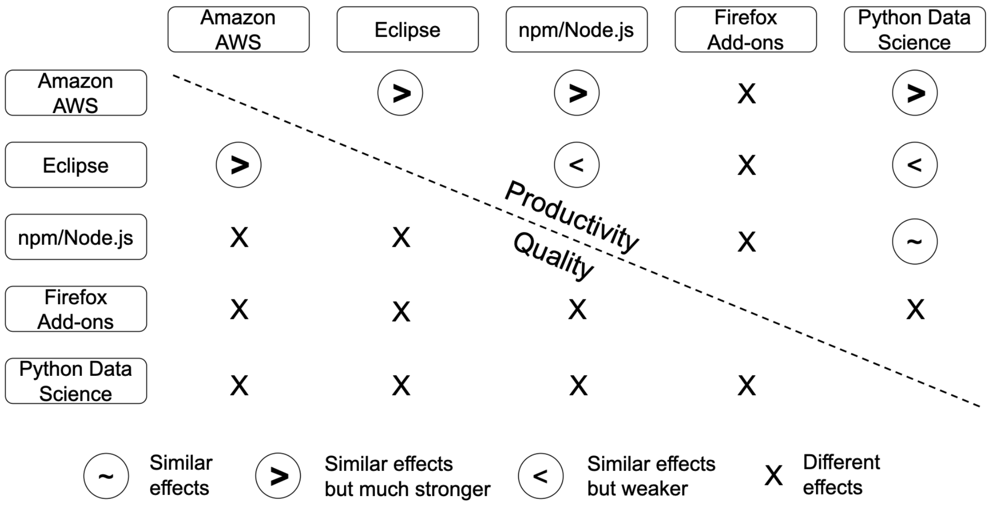


Figure 2.10: The pairwise similarities and differences between the relationships represented by the regression models.

Finding Summary: Similarity and Difference

According to the above results, there exist mixed relationships between effort allocations and project outcomes across target ecosystems. To clearly organize and summarize findings in an easy-to-interpret manner, here defines similar effects for two ecosystems (A and B with effect sizes of f_A and f_B) if (1) the significant *Independent Variables* are the same; (2) the coefficients of the significant variables have the same plus or minus signs; and (3) the differences of their models' effective size are within the range of 0.02, i.e., $|f_A - f_B| \leq 0.02$. If (1) and (2) were satisfied but (3) was not, here further defines "similar effects but much stronger" if the difference of effect size is greater than 0.06, i.e., $|f_A - f_B| > 0.06$; and "similar effects but stronger" if $0.02 < |f_A - f_B| \leq 0.06$. The definitions of "similar effects but much weaker" and "similar effects but weaker" are vice versa. If either (1) or (2) was not satisfied, A and B's relationships were "different effects". Following these defined notions, figure 2.10 summarizes these results for improved readability. The figure's top-right part describes the project productivity; the bottom left is about the project quality. The comparisons are from row to column.

There are some patterns that can be found in Figure 2.10. First, the relationships are consistent for project productivity especially the direction of the correlation, though they vary in degrees of effect sizes. Second, these relationships are completely diverse and ecosystem-specific for project quality. Third, elite developers' effort allocations in *Firefox-Addons* could not predict either project outcome indicator.

Answers to $\mathbf{RQ_{2.5}}$. Based on the above summary, $\mathbf{RQ_{2.5}}$ can be answered as follows:

Elite developers' effort allocations can significantly predict project outcomes for most of the targeted ecosystems. However, these relationships of project quality vary across different ecosystems:

1. **Project Productivity**: Efforts in communicative and supportive activities are generally negatively associated with the project productivity for all ecosystems except *Firefox Add-ons*. However, there are differences regarding the strengths of the effects.

2. **Project Quality**: The relationships are diverse. For *Amazon AWS*, *npm/Node.js*, and *Firefox Add-ons*, the effects are minimal. For *Eclipse* and *Python Data Science*, efforts in organizational activities significantly predict project quality; efforts in communicative activities are significant for *Python Data Science* but not for *Eclipse*.

Chapter 3

Small-scale Automation for Software Development

Open Source Software (OSS) projects are increasingly complex in scale, requiring practitioners' efforts in many tedious housekeeping tasks. Often, practitioners leverage software agents (aka bots) to automate their routine workflows and maintain their projects' efficiency and effectiveness. Relatedly, many OSS stakeholders, including practitioners and researchers, have invested significant resources to improve state-of-the-art bot techniques, resulting in numerous bots covering every aspect of modern software development practices, including technical and social aspects. Given this advance in practice, I seek to empirically identify how OSS projects adopt bot services from diverse selections. This empirical research examines bot applications in the most popular OSS repositories on GitHub and a follow-up interviewing study describes how bots integrate into developers' workflow.

3.1 The Usage of SE Bots on GitHub

Nowadays, mainstream Open Source Software (OSS) development often involves a huge base of contributors, users, and other stakeholders, which is afforded by various online social coding platforms, e.g., GitHub [64]. At the same time, the fast ongoing penetration of the CI/CD model poses urgent demands on rapid iterations. Facing the increasing pressures from managing a large number of contributors and their contributions, and the ever-faster pace in CI/CD, some small-scale automation techniques have found their way into OSS projects' engineering practices.

Software agents for Software Engineering (SE) tasks, i.e., devbots, SE bots, or just bots for short, are precisely such automation techniques that have the potential to automate many aspects in the process of building software. Various bots have been postulated for improving developers' workflow with better productivity and work-life balance [116]. Many of these software agents and bots are able to automate repetitive and routine tasks, such as managing Issues and Pull Requests, building, and executing test suites, and so on [103, 180]. Thus, they could often substitute for human labor, and developers may focus on more innovative and technical tasks. While these bots have saved substantial human efforts, they also have exhibited substantial limitations, for instance, not being designed for *smart* tasks and lacking interactivity or being disruptive during human-bot communication [51, 103].

Bots will inevitably be adopted by more OSS projects and also more closed-source commercial projects in the foreseeable future. Since the continuous development of automated software engineering and thanks to the continuous efforts of practitioners and researchers, the current bot market offers a wide range of software engineering services for OSS maintainers to shop around [97]. However, we know little about how practitioners select, adopt, and deploy these bots, especially among the most popular and socially successful OSS repositories. Without this stream of knowledge, we are not able to precisely identify the challenges of applying these bots, and ultimately optimize the workflow with bots. In this study, I start to bridge the gap by empirically examining the bots adopted by popular OSS projects on GitHub, the largest online social coding platform. The main research question that I intend to answer is,

RQ$_{\mathbf{3.1}}$ *What is the usage of small-scale automation, aka bot, in popular OSS development repositories?*

For this research question, I combined prior automated bot detection systems, validation through manual identifications, and bot data extraction. The study described in this chapter leverages an existing text-based method of automatic bot identification, which has been proven effective with GitHub data. Moreover, the other two researchers and I created a protocol for manually validating detection results, and identifying additional bots in OSS repositories with a multi-tier identification process [15]. For all identified bots, we extracted their information particularly on functionality, availability and deployment costs, etc. Additionally, this chapter includes a compiled list of bots that have been used in these 1,000 popular projects.

Analyses of these bot services show a remarkable adoption rate (61%) among popular GitHub OSS projects compared with prior studies [180]. However, the application scenarios of these bots are still limited: many of them are combinations of simple automation features triggered by specific project events and perform designated tasks through a rule-based mechanism. In addition, over 69% of bots remain private, without transparently sharing their implementation, nor becoming accessible outside projects or the organizational ecosystem.

3.1.1 Bot Identification Study Design

This study extracted activity traces of bot accounts and apps from GitHub repositories and identified the bots through existing automatic bot detection, and validate its results through manual review. First, to accelerate the process of automatically identifying bot activities from GitHub repositories, this study employs BoDeGHa, a machine learning classifier using language patterns and features of comments to detect bot activities [65]. Second, for data reliability, the other two researchers and I manually examined the automatically classified bot services by creating and following a manual identification protocol and further reviewed their activities. The overall research process can be described in Figure 3.1.

Sampled Repositories

The sampling process of target repositories is based on three considerations. The first consideration is the number of GitHub *star* as an indicator of popularity and also associated with stages of project evolution [21]. Sampling from most starred repositories further helps to avoid many perils in mining repositories, e.g., projects that received minimal community participation [83]. Second, similar reasoning as previous OSS studies in Chapter 2, I ensured that the sample only includes software development projects using the Pull Request model, which excluded repositories of storage, knowledge sharing, and other non-SE use. This consideration also promises that the automatic classifier may process substantial bot activities for identification. Finally, the sample size was set at 1,000, which is due to 1) a long tail distribution of contributing activities across GitHub repositories, i.e., only a marginal proportion of repositories is receiving active contributions; and 2) the cost of manual project selection, and later bot identification and functionality reviews.

Based on the above considerations, the complete list of sample repositories is included at `https://tinyurl.com/33392bp3`.

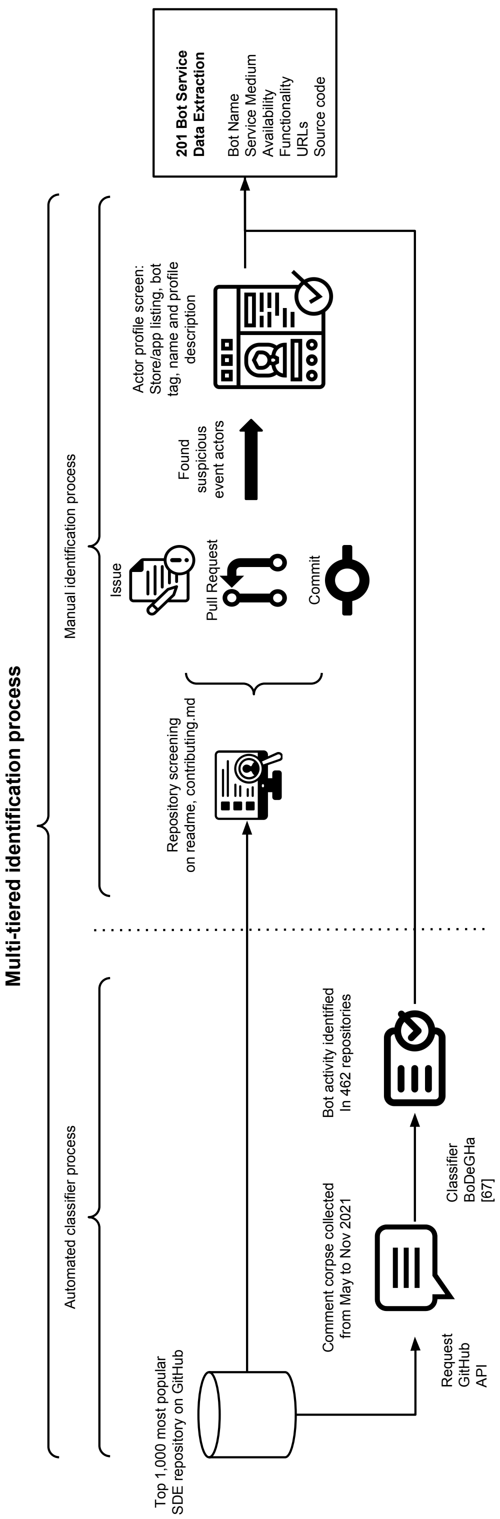

Multi-tiered identification process
Automated classifier process
Manual identification process
Top 1,000 most popular SDE repository on GitHub
Request GitHub API
Comment corpse collected from May to Nov 2021
Classifier BoDeGHa [67]
Bot activity identified In 462 repositories
Repository screening on readme, contributing.md
Issue
Pull Request
Commit
Found suspicious event actors
Actor profile screen: Store/app listing, bot tag, name and profile description
201 Bot Service Data Extraction
Bot Name
Service Medium
Availability
Functionality
URLs
Source code

Figure 3.1: GitHub bot identification study process

Automatic Bot Identification Process

Mainly to identify machine-generated traces from countless social and technical activity events in OSS repositories, I first applied a pre-trained machine learning (ML) classifier, BoDeGHa, to shorten a list of suspicious event actors exerting machine behaviors [65].

This machine learning classifier mainly applies the main rationale that the comments of a bot have fewer patterns and are more repetitive than humans since their responses are event-triggered with pre-programmed rule-based designs. Therefore, the classifier's model applies language features such as text distance, text patterns, pattern inequalities, and the number of comments. By applying this pre-trained classifier, I was able to select an initial list of bot actors.

This classifier collected comment data and executed its classification in November 2021. To automate the process, a set of Python scripts looped through all sampled repositories, and they follow the default setting of this classifier by collecting 6 months of repository comments. Its results were exported and stored in local `cvs` files.

Although this classifier provided an initial list of bots, there are some limitations while examining the results. First, the classifier cannot handle comments from humans by using other languages. Secondly, many bots cannot be found in repository comments alone. For instance, several bots post an issue or pull request in the repository by reminding contributors of security updates. Finally, prior research suggests that bot or machine-generated content identification requires a multi-tier process to achieve optimal reliability [15].

Multi-Tier Manual Bot Identification Process

This study aims to provide improved bot identification results based on binary classifications in social or social-technical platforms, and therefore, as prior research suggested, several types of supporting data are necessary for more accurate results [15, 85]. As recommended in these studies, a bot can be more accurately identified through a multi-tier identification process, typically including examining its "social network" of its following and follower relationships, its comment and other generated text, account metadata, and activity patterns.

This study employs a three-tier identification based on GitHub data, including repository comment text, account meta-data, and event patterns of its timeline activities. Particularly, many bot-based event actors on GitHub do not have a "social" network as other social media platforms, and moreover, the social network features in GitHub are not as significantly changing user behaviors as they were on other social platforms. Furthermore, the chance of a GitHub user who is a real human but not having any followers or following anyone, is significantly higher than other social-native platforms, e.g., Twitter and Facebook, due to GitHub's technical nature [42]. Therefore, this study argues that social network is not a reliable tier of data for bot identification, and applying this data tier would neglect the characteristics of this social-technical platform. However, other generic data tiers still apply for the context of GitHub.

The remainder of this section introduces how the above multi-tier classification gets operationalized and provides a protocol for replicating this process among our research team. The other two researchers and I followed this protocol to validate automated results and identify additional bots. Given a target repository, there are several features to manually spot applications of bot services, including repository `readme`, Commit list, Issue list, Pull Request list, and Release list, as bots usually provide services for these tasks.

The first screening overall is to examine the **repository readme**s under a target repository root directory, including manually reviewing the content of `readme.md` and `contributing.md` files. According to our observation, substantial well-maintained and popular repositories provide these statements of whether the repository has employed any bots in their established workflow, for instance:

```
 ...for this repository, the contributing.md file demonstrates that
a CLA bot would interact with the incoming Pull Requester to collect
signatures.
```

In this case, its contributors have clearly indicated that this repository employs a `CLA` bot to solicit signatures from external contributions. Moreover, contributors often provide information about other bot applications such as Issue/Pull Request formatting, contribution acknowledgment, and so on.

Since bots are triggered by certain repository events, checking the most critical OSS development events and their *repository event actors* leads researchers to find bot services under this mechanism. GitHub's event timeline of Issue, Pull Request, Commit, and Release provides researchers access to identify bot services. Without spending too much effort on historical events, I focused on recent ones which also reflects whether the repository is running these bot services now. Particularly for each type of event,

- **Commit**: Review the latest 20 Commits, and verify whether each Commit event actor, including the commenter of each commit, is a bot service.
- **Issue**: Review the latest 10 open and 10 closed Issues in the Issue management system. Verify if each Issue event actor and Issue commenter is a bot service.
- **Pull Request**: Review the latest 10 open and 10 closed Pull Requests in the Pull Request management system. Verify if each Pull Request event actor and commenter is a bot service.

From the above locations, when we found an event actor that was suspiciously repeating activity patterns or demonstrated that the content was automatically generated, we applied this multi-tier identification by first reviewing its account metadata such as GitHub profile or GitHub Marketplace/App listings, and examining its naming and description; second, reviewing the content of this GitHub account's recent comments and activities; finally, for accounts that did not display their recent contributions, collected their activity records through GitHub API.

Additionally, here are several extra indicators that help to confirm whether an event actor is a bot:

- Being listed in GitHub Apps, Marketplace, and Actions.
- Having a `bot` tag next to the actor name.
- Having keywords in its description and name that indicates bots.

For instance, we can determine if there is a description claiming that this is a bot for some repository, a demonstration of the comment or description is "automatically generated," or bot account name keywords include `bot, ci, cla, auto, logic, code, io,` and `assist` [65]. However, we were additionally careful to determine when found any other human-like behaviors in its profile, such as following other accounts and displaying personal email addresses.

Another researcher and I discussed and generated the above protocol for examining and identifying bot services, and then tested this protocol on a random sub-sample of 100 repositories. This initial test achieved an inter-rater reliability of Cohen's $\kappa = 0.65$, which suggests a substantial agreement. Differences in identification were resolved during a discussion session and led to an update of this protocol. Finally, three researchers followed the updated protocol for identifying and labeling bots in the entire sample.

For each of the accounts that have been identified as a bot, we record the following content, including:

- **Bot account name**: the profile, app, or service names of the event actor
- **Service Medium**: whether the bot is found in GitHub (private) App or it has a profile
- **Availability**: whether the bot service is available to the public or private organization, free or commercialized.
- **Functionality Notes**: Notes about performed tasks of the bot, for example, the committed code is for updating dependencies according to its commit message, the issue closes due to staling for too long, the Pull Request is automatically generated to merge code from the testing branch, and label the issue due to no one is investigating (an untriaged label).
- **URLs**: important and related URLs to the account profile page or GitHub App.
- **Open-sourced**: whether there is any indication that the bot is transparently open-sourced, from its profile and App page.

Several sets of bot accounts/actors provided the same service as one application or employed different actor names between various bot versions. For example, `codecov-io` and `codecov-commenter` worked together for posting results and feedback about test cases execution; and `dependabot-preview`, `dependabot`, and `dependabot-io` are mainly the same bot in different historical versions according to its documentation. To avoid repetitive counting, we decided to merge these bot sets into one bot service.

3.1.2 The State-of-the-Art of SE Bots on GitHub

Among the 1,000 sampled repositories, `BoDeGHa` identified bot activities in 462 repositories. Following the steps described in Section 3.1.1, this multi-tier manual identification process

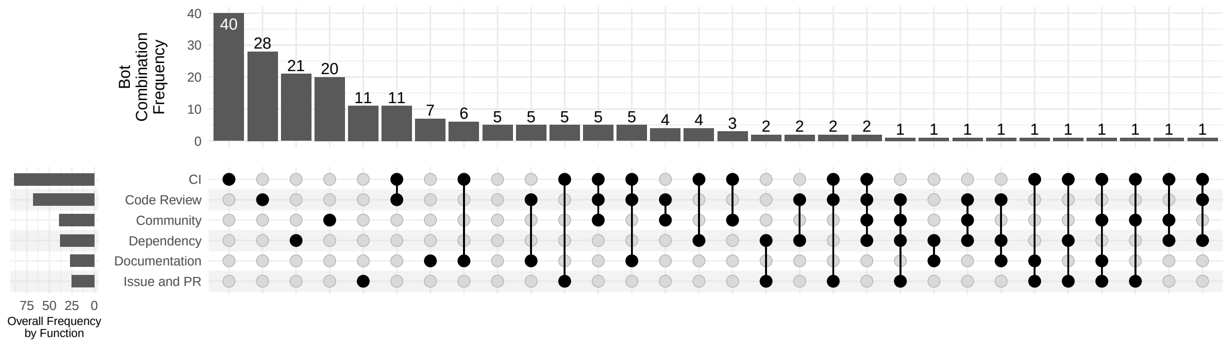


Figure 3.2: Frequency of bot function combinations and overall bot function frequency (in sampled projects)

found additional repositories apply more bots, i.e., 613 sampled repositories have employed 201 distinctive bot services. Three of found bots were reported as deprecated or had stopped services. Particularly, all results of this study are included in a data set of

- A list of popular software development repositories that have employed bots in the CI/CD practice;
- Descriptions of bot accounts, apps, and services employed by these repositories.

According to this data set, this study has found a significant number (61%) of popular repositories in this sample have applied bots in assisting their daily activities. In the following sections, I will introduce the found bots' functionalities, their combinations, and the most popular bots in the sample.

Bot Function Categories and Combinations

In addition, based on our collected data on bot functionalities, there are six main categories of bots' functions that substantially overlapped with a prior study [180]. However, many bots found in this study provide combinations of functions, and I will explain them later in the section. Let us first look at the major function categories below.

- **CI Tasks Assistance**: the service automates the various task for CI/CD, including managing branches and releases (`release-drafter, semantic-release-bot`), pushing commits and merging approved/passed pull requests (BORS, `lgtm-com`, and `repo-ranger`), and automating building, testing, and deployment (`vercel`, `buildbot`, and `bors`).

- **Issue and PR Management**: the service automates the management of Issue and Pull Request tracking systems, including labeling issues and pull requests, and cleaning inactive/invalid ones (`mary-poppins, carsonbot, support`), and requesting or formatting issue and pull request content with checklists (`issue-check, request-info, vue-bot`).

- **Code Review Assistance**: the services provide assistance in the code review process, including performing static and dynamic analyses (`sourcery-ai, sourcelevel-bot, codecov`), summarizing changeset and visualizing difference (`changeset-bot, ecma262-compare-bot, sizebot`), and assigning code reviewer (`googlebot, pullapprove, vscode-triage-bot`).

- **Dependency and Security**: the service periodically checks and ensures that the repository's dependency is update-to-date, and also detects and reports security issues (`dependabot, depfu, renovate`).

- **Developer and User Community Support**: the service assists the management of the community through collecting contributor license agreements (CLA), acknowledging contributors (`allcontributors`), managing contributor permissions (`meeseeksdev`), and auto-commenting for welcoming or feedback (`MIRAI-bot, welcome`).

- **Documentation Generation**: the service assists generating end-user and developer documentation (`decdocs-bot, weekly-digest, rc-publisher`).

These application scenarios and functions of adopted bot services are similar to prior research [156, 180], but we adopt a coarse-grained classification for the convenience of identifying multi-functioning bots (i.e., butler bots in this study). According to the list of combinations (see the lower left corner in Figure 3.2), the most prevalent functions are automating CI tasks and community management, and the least occurring one is Issue and PR management. Five bot services cannot be included in any of the above function categories. Combinations of functions are less prevalent according to the number of distinctive bot services. The most prevalent combination is able to support CI and Code Review tasks simultaneously.

Many bots have specialized in multiple tasks and become a *butler* for the repository, and therefore the combinations of bot functions have a wide range of variety. For example, `repokitteh` performs multiple tasks, including checking the format of the pull request, automating tests on the pull request, assigning users to issue and labeling issues, and also merging pull requests that passed the auto-tests. Some organizations even have customized their private *butler* bot overseeing their CI/CD practices, e.g., `Googlebot` and `facebook-github-bot` in Table 3.1. Due to repository customization, various bots emerged with a unique combination of functionalities (see the upper corner in Figure 3.2).

SE Bot Services in Most Popular Repositories

In this section of findings, I list the most prevalent bot services based on appearance in the number of projects in Table 3.1, and the most frequently appeared one is the `dependabot` which periodically checks the dependence of the source codes for a repository, and issues a Pull Request when the imported packages are out-of-date. `Renovate`, ranks at 6th in this list, also provides the same set of services as `dependabot`. The second most appeared bot is the `stale` bot, which set up timers for Issue and Pull Requests that are no longer active. It reminds developers to take action with these open Issues or Pull Requests after a customized time. `codecov` (3rd), `coveralls` (9th), and `vercel` (10th) are all CI assistants. These bots help to automate building, testing, and presenting the results of changes and tests. `Googlebot` (4th) and `facebook-github-bot` (7th) are two ecosystem-specific bots, which are meant to service like repository *butlers* based on the standards of their organizational ecosystem. These two bots provide a range of assembled and customized functions. For instance, `Googlebot` has more emphasis on triaging Pull Requests and Issues, while the `facebook-github-bot` mainly focuses on CI tasks. Finally, `CLAassistant` (5th) is an open-source-specific bot, which collects external contributors' signatures on the contributor license agreement of the repository.

There are a few common patterns for the above services. Noticeably, all these popular services provide a native setup option with GitHub Apps, and may additionally provide services through GitHub Actions except the `facebook-github-bot` only employing an authorized account. Moreover, Pull Request comment is the main communication channel for all popular bots, besides the Stale bot which is also specialized in Issues. Further, these services provide a free installment option to the public, though two services (`Googlebot` and `facebook-github-bot`) are only available to their organizational ecosystems.

3.1.3 Finding Summary

This study provided a list of repositories that have employed bot assistance in their CI/CD practice, described their adopted bot services and identified the most popular bots among these repositories. According to this bot identification study, bot services on GitHub provide a wide range of functionalities with distinct combinations according to their repository focuses. Moreover, the most widely adopted bots are equipped with easy installation, assisted Pull-Request-related tasks, and are customized in supporting their ecosystems.

Table 3.1: Top ten of most prevalent bot services among popular OSS development projects

Bot Service	**Service Medium**	**Interactions**	**Cost for Developers**	**Functionalities**	**Code Availability**	**Service Availability**	**Creation Framework**	**#Projects**
Dependabot	GitHub repository security setting	Pull request and Pull request comment	Free	Update dependencies of a repository from automatically generated pull request.	Yes	Public	n.a.	171
stale	GitHub Apps	Issue and pull request comment	Free	Label inactive issues and pull requests as "stale", and close them if remain inactive.	Yes	Public	ProBot	100
codecov	GitHub Apps GitHub Actions	Pull request comment	Free and paid	Report the test coverage of a pull request.	No	Public	n.a.	89
Googlebot	GitHub Apps Authorized account	Pull request comment	Private	Check and collect CLA, and label the issue or pull request with its CLA status. Assign developers to issue and pull requests. Collect user feedback after the issue closes.	Yes	Organizational	ProBot	51
CLAassistant	GitHub Apps GitHub Actions	Pull request comment	Free	Comments on pull requests to ask contributor to sign CLA.	Yes	Public	n.a.	46
Renovate	GitHub Apps	Pull request and Pull request comment	Free	Update dependencies of a repository from automatically generated pull request.	Yes	Public	n.a.	34
facebook-github -bot	Authorized account	Pull request comment	Private	Comments on a pull request to ask the contributor to sign CLA. Commit and push changes. Summarize changes of a pull request and execute auto-tests. Label CI status on the pull request.	No	Organizational	n.a.	30
coveralls	GitHub Apps GitHub Actions	Pull request comment	Free and paid	Report the test coverage of a pull request	No	Public	n.a.	26
vercel	GitHub Apps GitHub Actions	Pull request comment	Free and paid	Automate CI for building, testing, and deployment for front-end applications. Comment on the pull request includes a URL of the preview of the deployed changes.	No	Public	n.a.	20

We do not include GitHub Action as a typical bot service in this study, because it is usually powered by existing or customized automation.

3.2 Bot-Assisted Workflows for Individual Practitioners

With the development of automation technologies, software production urged itself to also keep up with the progression of automation. Many automation services have gradually made up each aspect of the daily practice of software development, and with a unified goal to enhance our productivity [59, 180]. For instance, these automation services help set up CI/CD pipelines, enable bot-assisted workflows, and more recently, pair up with the AI programming assistant in low-level source code production, etc [135]. These services have significant impacts on improving software developers' work-life balance and also enhancing engineering productivity.

While each above domain exerts its unique importance in automating software engineering, bots have become ubiquitous in practice and assisted various perspectives of our daily software engineering tasks, including technical and non-technical ones. Moreover, my previous studies described in Chapter 2 have demonstrated the substantial role of non-technical tasks in developing software and maintaining OSS projects, and particularly how these tasks are associated with productivity and other technical outcomes [172]. Therefore, it is necessary for us to provide an in-depth view of how the latest SE bot assisted these dimensions of software engineering activities, and further, we may utilize and improve the designs of these SE bots.

Built upon other research and practice efforts in SE bots, this study particularly aims to reveal a full picture of the current bot-assisted workflow according to the perceptions of software developers themselves. Thus, we may understand practitioners' experience of using SE bots and expectations of bot development. As described in the previous section of this chapter, bots have been widely employed in assisting OSS development, especially for most socially successful and contribution-intense projects. While there are many bots deployed in practice and numerous other choices on the shelf, we know little about how developers integrate these bots into their daily workflow and how they perceive the values that are brought by these SE bots. Therefore, I intend to answer the following research questions about their novel workflow with bot assistance,

RQ$_{3.2}$: *How do developers integrate bot services into their daily workflow of developing software?*

and further, identify current challenges and provide further guidance on bot development,

RQ$_{3.3}$: *What are the challenges and expectations of the current SE bot from the perspective of expert developers?*

To answer these two research questions, I employed a purposive sampling process to solicit participants and conducted semi-structured interviews when collecting developer feedback. Email invitations, flyers, or broadcast messages at organizations' public channels were delivered to prospective participants respectively. According to their completion of intent forms, a purposive sampling process based on repository popularity and bot deployment was applied when selecting participants from various repositories. Then developers' feedback was collected through a semi-structured interview process with 10 pre-defined questions under three main question sections. I analyzed their feedback through an open coding procedure by annotating raw interview transcripts and conducted thematic analyses to extract meaningful and in-depth findings. This study was categorized as exempted according to UCI's IRB office.

The findings of this study suggest that software engineering (SE) bots have been immensely helpful to elite developers in both technical and non-technical activities. Nevertheless, participants reported that there is still room for improvement in terms of usability. The study therefore offers recommendations on how to allocate limited engineering and research resources to improve, with a focus on enhancing developers' workflows. Furthermore, the study highlights the potential for future SE bots to better support elite developers in managing their increasing workload resulting from rising demands.

3.2.1 Research Methods

Participant Solicitation and Sampling

Two streams of participant solicitations happen simultaneously with OSS developers and company-employed developers.

For recruiting OSS developers, this study reuses sampled repositories in the first study of this chapter. Based on a similar reason as I mentioned in Section 3.1.1, i.e., using GitHub stars as an indicator while sampling repositories ensure the activity level, I reuse the sample of top 1,000 starred repositories which can also cross-validate the prior data mining study. From this repository sample, my prior bot identification results in Section 3.1.2 verified a list of repositories with bot activities (from May to November 2021). Then I solicited elite developers of these repositories who were able to perform all types of activities and made decisions of deploying bots, and I identified elite developers on these repositories based on the history of critical issue activities, including merging, assigning, unassigning, etc. When elite developers were found, direct email invitations were sent to them individually. Prospective participants may sign up through intent forms created via Google Forms[1].

For recruiting developers who are mainly employed by software development companies, the first author leverages his personal professional network and reaches out to several organizations, including two large global software corporations and two mid-size software companies

[1]Google Forms: `https://www.google.com/forms/about/`

based in North America. When given permission, a soliciting message or flyer was posted in the organization's non-work related slack channel, and participants may sign up through the intent form (the entire intent form can be found in Appendix A.2).

Interview Study Design

To explore participant thoughts, feelings, and beliefs about SE bot and delve deeply into their daily workflow, this study employed semi-structured interviews [84]. There are three major sections in the interview to answer the two main research questions. The first section aims to understand the developers' routine workflow, and how the bot partially assisted their work. The second section explores their feelings about the current bots and their beliefs toward future bots. The last section was mainly about the logistics including signing up for receiving early publications and participating in follow-up studies.

Interview studies lasted between 25 to 40 minutes, and most completed around 25 minutes as estimated. All interviews were conducted online via the Zoom video conferencing software[2], due to the impact of the COVID-19 pandemic during 2021 and 2022. Participant information of this study can be found in Table 3.2. Particularly, participants from open-source projects were ones with write access to the repository, and participants who mainly contributed to commercial and closed-source projects were repository admins or technical leads. Interviewees were welcome to decline to answer any questions that they felt uncomfortable with, and they might exit the interview study at any point. Additionally, the interview protocol is attached in Appendix A.3.

All interviews were recorded through the built-in recording function provided by Zoom after obtaining participant consent. Moreover, the recordings were automatically transcribed in YuJa[3] video editing software, and manually corrected based on auto-generation results.

[2]Zoom: `https://zoom.us/`

[3]Yuja: `https://www.yuja.com/capabilities/video-editor/`

Table 3.2: Demographic details of interview participants

ID	Gender	Country	Most Recent Job Role	Organization/Project Type
P1	Male	Canada	Software Engineer	Commercial Open Source*
P2	Male	Canada	Support Engineer	Commercial Open Source
P3	Male	USA	Software Engineer	Commercial Software
P4	Female	USA	Data Engineer	Commercial Software
P5	Male	USA	Sr. Software Engineer	Commercial Software and Open Source
P6	Male	USA	Research Engineer	Open Source
P7	Male	USA	Sr. Software Engineer	Commercial Software

*: Commercial Open Source specifically refers to projects which aims at profiting and open-sourced their code.

After interviews were finished and transcripts corrected, two researchers used qualitative analysis, the coding technique, to iteratively identify common themes that emerged across the interviews and create a team codebook [108]. The coding schema was validated on one participant's transcript, and its coding achieved substantial agreement (Cohen's $\kappa = 0.66$). I first discuss these themes in the following sub-section.

3.2.2 Interview Findings

Overall Workflow and Notification

Interview results have shown that automation has a critical influence on the daily workflow of these developers. With the increasing integration of technical SE bots in the repository, automation has penetrated many aspects of the engineering process including test executions, deployment, and other development pipeline stages. There are two main approaches of SE bots integrating into a repository's workflow and maintenance: *proactive* and *reactive*.

Reactive bots are ones enhancing and assisting human tasks, let us use the pull request workflow as an example. Pull request is a widely-accepted basic unit of open-source technical contribution. For a well-maintained and automation-assisted OSS repository, initiating a pull request requires following a pre-defined format by filling in several fields. When external contributors submitted the pull requests, two automation services may intervene: pull request formatting and CLA collectors. Pull request formatting checks whether the target pull request follows a certain format, and the Contributor License Agreement (CLA) bot asks for contributors' signatures on the agreement such as releasing intellectual properties if they have not yet (P1 and P2). Particularly, there are repositories that enable visualization or text-based summary to assist by leaving bot-generated comments below that pull request (P3). If these checks pass, the pull request proceeds to build and test. SE bots such as `codecov` and `coverall` (see Table 3.1) build and generate a test report as additional review information assisted the code review process. Meanwhile, some repository leverage a pull request triage bot to label and assign the code reviewers for the convenience of backlog management (P6). Finally, deployment previews provided by bots such as `netlify`[4] and `Travis`[5] assisted code reviewers to assess the quality of the implementation in the pull requests (P1). Other assistive bots also proactively help developers' daily workflow, including auto-generating release drafts based on pull requests and commit messages, and welcoming first-time contributors for submitting issues, etc.

Proactive bots monitor the artifacts of repositories, update resources from internet, and remind or provide actionable measures to correct unwanted human behaviors. One of the most popular SE bots of this category is `depandabot`. The `depandabot` received positive feedback from developers (P1, P3, and P5) as it "provides significant technical values." The mechanism of `depandabot` is to scan source code headers and package dependencies in the repository, and when there is a critical release or update of these packages the `depandabot` initiates a pull request to update the source code. Though sometimes the bot was "un-necessarily sensitive," this feature helps developers to maintain the repository dependencies up-to-date and mitigate security concerns. Another active example is the `stale` bot (P1,

[4]netlify: `https://www.netlify.com/`
[5]Travis: `https://www.travis-ci.com/`

P3, and P4). This bot monitors all repository issues and pull requests, and when these items have not received any actions (including comments and events) for a certain period of time, the bot labels them and/or comments with a "stale" indicator so that reminds developers to take actions with inactive backlog items or requests. This bot and its similar alternatives (`marypoppins` bot, etc) helped developers to manage repository backlogs, especially with a growing community (P2). Other proactive bots were also mentioned in developers' daily workflow, for instance, activity summary (P7) and community acknowledgment (P2 and P5).

Besides their bot-assisted workflow, another finding worth noting is that developers often rely on GitHub's notification system to keep up with the latest of their artifact and contributor/user community. As elite developers of a repository, they receive all types of notifications by default: for example, being assigned to an issue or pull request, opening a pull request, issue, or created a team discussion post, commenting on any of these threads, subscribing to a thread, changing the state of a thread (issue events) and finally having username `@mentioned`[6]. These notifications provide comprehensive views about the technical and social updates of a repository. In addition to a variety of updates, elite developers heavily relied on notifications to provide timely support to the community. Quick responses on threads to their contributor and user community are regarded as an important trait to maintain a positive relationship with their users (P1 and P5), and also enhance developer reputation and publicity. For instance, P2 commented on supporting their community, "you often [need to] go to Slack channel and Google Groups, and that's like a really active community. So you're the person in a way going through the best you can and interacting with community members." Similarly, as a founder of their software project and pushing this project into its commercial path, P1 mentioned, "**INSERT A QUOTE HERE.**" While all OSS participants are full-time employees somewhere other than their OSS projects, sometimes they do not have full effort in planning and managing their projects. Therefore, community

[6]About notifications: `https://docs.github.com/en/account-and-profile/managing-subscriptions-and-notifications-on-github/setting-up-notifications/about-notifications`

requests have been a mixed blessing to the projects: on one hand, developers have a backlog from the community to work on (P1, P5); and on the other hand, they might feel stressed when the latest requests come in (P1, P2, and P5). As a result, checking notifications has a high priority in developers' workflow, but checking notifications sometimes can be a stressful experience.

Benefits of SE Bots

The benefits of using SE bots are tri-fold. First of all, SE bots provide substantial technical values according to many of the participants (P1, P2, P3, P5, and P7). As mentioned above, SE bots automated many aspects of their workflows in proactive and reactive ways. These bots active in the CI/CD process have saved human efforts of building and testing, and provide additional information to assess the code review process. "*...One, two, three...I guess we now have four checks in CI now,* [*and*] *I don't think I would remember to run them all every time* [*not if the bot would run them for me*](P5)." As this participant commented, the automated workflow helped remind or directly executed checks for developers in this ever fast pacing development. These tasks help developers, in their words, "provided technical values (P5)" to their daily development workflow, i.e., significantly improve their productivity working with the CI/CD pipeline and enhance merged codebase quality via various automated quality assurance measures.

The second benefit of SE bots is that they assisted in providing in-time support for the community. As I reported in Chapter 2, elite developers often have to spend considerable effort to support and organize their community, especially when the community grows over time. While elite developers underwent these increasing supportive and organizational responsibilities, they have been provided with automation support for these related tasks, e.g., verifying whether the contributor has signed CLA, welcoming them as first-time contributors, and requesting contributors to follow contributing guidelines and repository code of conduct, etc. These types of bot assistance serve as a buffer between developers' community support work and other activities. With this automated in-time help from SE bots, elite developers will not have to always monitor community updates and focus on their creative work or other things in life. For instance, P1 commented, "*...not having to watch my phone's notification would feel good.*"

Finally, while automation helps developers with many aspects of software development, these SE bots also help developers to focus on creative tasks or other things in their lives. As mentioned above, automation assistance helps developers alleviate their mental load of various engineering tasks. Particularly, OSS elite developers often worry about various responsibilities for their projects, "*...[contributing to* OSS] *not like what I work during the daytime, I need to take care of everything in this project.* (P5)"

Challenges and Expectations of SE Bots

Confirming with prior SE bots studies [103, 180], the two substantial challenges of using SE bots were excessive notifications and limited interactivity, mentioned by elite developer participants.

First, elite developers have complained that SE bots have produced additional notifications, especially proactive bots during irregular development times. Different from reactive bots giving immediate feedback during work sessions of CI/CD pipelines, proactive bots such as `stale` and `weekly-digest` might post new threads beyond core development hours. Many elite developers such as P1 and P5 in this study employ a phone App to keep monitoring the repository. However, additional notifications either from bots or other spam may annoy and stress developers.

The second major challenge is the interactivity of SE bots. In the previous study of this chapter, I mentioned that popular SE bots employ rule-based design, and as a consequence, they would make pre-defined reactions to certain repository events or thread content. These restricted ways of interaction can be effective and direct with experienced developers who required additional information in CI/CD pipelines, "*I think the bots that we've used haven't been interactive. I think they've all just been kind of dump*[*ing*] *a lot of information* (P3)." Although developers have to extract useful information from what bots provided, its information has significant value to their technical practice. Nevertheless, lacking interactivity can be troublesome when working with novices and users from communities. For instance, as a community support engineer, P2 commented due to lacking interactivity, "...*we want to make sure the community knows what's going on about the project, or how they felt about it. And I have to do it manually, there* [*were*] *no quick automated ways.*" Compared with more technical CI/CD bots, community support bots require certain conversational capabilities to provide contributor guidelines and collect feedback.

Based on elite developers' experience with SE bots, they talked freely about their expectations of the functional and non-functional features of future bots. First of all, elite developers re-emphasized the importance of providing technical values as current SE bots did. For instance, P2 directly commented that current automation features can be integrated, "*improve the functionality you want to see, and sum up to the current bot.*" Particularly, participants asked for more technical functions including programming language support for code coverage (P3), Issue-Pullrequest linkage (P2), and JIRA issue tracking integration (P4), etc.

The last item on the wish list was following the OSS spirit when developing these bots, "...*if you*[*r project is*] *free and you don't have to pay. I think the payment model is, you have to pay if you want your repo to be private.* (P3)" Moreover, as another participant said, two large-scale organizations that he involved choose to withhold the implementation details of their SE bots, which was due to security and privacy reasons. Therefore, he had wished that there would be open-source reproduced versions of these bots, "*I do tend to feel like* [*their bots*] *are really high-quality and they could do a lot of different things, so it'd be cool to see* *that being reproduced open-source more and more.* (P2)"

Answers to RQs

According to my analyses of the first section of the interview, we can answer the first $\mathbf{RQ_{3.2}}$ about the developer's bot-assisted workflow as follows:

Elite developers have substantial integration of SE bots into their daily development workflow. Particularly, SE bots assist developers through proactive and reactive approaches and automate various aspects of their work.

As answering the first question helps us understand elite developers' bot-assisted workflow, these developers have expressed their experience while having these bots automating their workflow in the following interview sections. The question $\mathbf{RQ_{3.3}}$ can be answered with analyses of the latter two sections of interviews,

Elite developers have acknowledged that SE bots provided substantial technical value in their engineering workflow, support for their community, and concentration on other tasks. However, interactivity and additional notifications have been two major challenges to elite developers while using bots. For their expectations of future SE bots, elite developers have emphasized their wishes for simplicity in design and use, technical values of automating mechanical tasks, and OSS spirits.

Chapter 4

Bot Management Console

In the final study of this dissertation, I introduce a management console for bots and small-scale automation functions for GitHub based software development. The proposed prototype aims to resolve or at least alleviate many limitations of SE bots, and follow the implications discussed in previous studies (see findings of above Section 3.1.2 and 3.2). This console will support developers in customizing detailed configurations while integrating automation into their workflow. This section includes a list of design requirements and specifications derived from the results of the previous two studies and current SE bot literature. Finally, this section provides a detailed evaluation plan for the proposed console through .

4.1 Management Console for SE Bots

The practice of software engineering evolves at an unprecedented pace nowadays, and it has become more reliably automated with the latest technologies proliferating. For instance, AI-powered applications of pair-programming assistance (e.g., copilot) and on-demand knowledge inquiry (e.g., ChatGPT), and infrastructure advances such as employing

CI/CD pipelines and SE bots, all these technologies accelerate and enhance our process of crafting modern software artifacts at every level of scale.

As these technologies have penetrated into many organizations' daily engineering processes, SE bots, aka devbots or just bots for short, have been the central interface of human-AI interactions [103]. SE-bot-assisted engineering workflows could significantly simplify development steps, create buffers of context switching, reduce human efforts by automating task execution and delegation, etc [180]. However, prior research and my previous chapters of studies have indicated that there were several critical drawbacks of SE bots in the current software engineering practice. With these obstacles hanging in there, developers, especially elite developers in the OSS projects, have to bear many usability issues, namely excessive notifications and interrupts [156], opaque processes of (re-)implementing SE bots in silos, and non-customize-able automation combinations [174]. Although developers have integrated a lot of SE bots for their engineering work, these above issues have been reported to affect bots' usability and extensibility repeatedly [103, 156, 174, 180].

Building upon existing work, software engineering researchers have postulated many solutions to improve current SE bots, but their work often focuses on improving single SE bots or enhancing some key aspects of bot-human interactions [181]. Therefore, this chapter of this dissertation mainly tries to answer the research question,

RQ$_{4.1}$ *How to design and evaluate a bot framework that supports assembling current automation features and provides management customization?*

To answer this overall research question, I proposed a novel solution by implementing a bot management console with an overall goal to alleviate these issues of SE bots. First, by summarizing prior findings of SE bot issues and my study results in previous chapters, I solicited design guidelines for integrating bot services under a management framework. Moreover, based on my empirical findings from popular OSS repositories and results of interview studies with elite developers, secondly, I derive a set of design specifications for implementing this bot management console prototype. Finally, this prototype was evaluated through a simulated deployment in an agent-based model. The evaluation tested the effectiveness of reducing noises generated by proactive repository monitoring from these SE bots.

Following this proposal and design specifications, this study implemented a prototype of a bot management console with Python-based data processing services and a ReactJS web application. The high-level architecture of this management console largely follows a classic event-subscriber structure [115]. When presenting this management console prototype, I introduce its three main views including automation selection, control panel, and activity summary, and present their use case scenarios respectively. To test its implementation and effectiveness towards major repositories, I simulated deployment on several Python OSS projects of mainstream data science frameworks based on these projects' longitudinal event data (sampled and collected in Chapter 2). Based on the longitudinal event data from five major Python data science projects, my preliminary evaluation results have shown positive results: the management console could moderate the flow of notifications by triaging them to both noticing mechanisms according to the automation's workflow integration mechanism.

This chapter's organization is as the followings. I first present the design guidelines of a bot management console based on prior literature and my previous studies. Then I will illustrate the design specifications including the principal use case scenarios of this console. Finally, to validate its mechanism for reducing bot interruptions, the following section introduces the evaluation based on simulated deployment.

4.2 Design Rationales

4.2.1 Design Guidelines and Requirements

The overall goal of this prototype is to design an infrastructure foundation across different types of SE bots with a central level of control that oversees notifications, activities, and functionalities of bots. To provide comprehensive design guidelines for such a list, this study employs three major sources to elicit the design guidelines [26, p. 2], evidence from empirical studies (Chapter 2 and Chapter 3), predictions from existing theories (existing literature of bots), and evidence gathered through engineering experience (Section 3.2).

For clarity, Table 4.1 illustrates the evidence base which supports these design guidelines (DG). The column on design guidelines illustrates the content of a specific guide for implementing a managerial infrastructure for bots. The Empirical Evidence column includes results of answering empirical investigation questions $\mathbf{RQ_{2.1}}$, $\mathbf{RQ_{2.2}}$, $\mathbf{RQ_{2.3}}$, $\mathbf{RQ_{2.4}}$, and $\mathbf{RQ_{2.5}}$. Engineering Experience can be evident by interview study in the second part of Chapter 3, which includes answering $\mathbf{RQ_{3.2}}$ and $\mathbf{RQ_{3.3}}$. Finally, existing theories and literature support can be found at various scientific venues including CSCW, CHI, IEEE Software, BotSE workshops, etc. Particularly, the publications were screened at the BotSE research repository [3]. These six main design guidelines provide cornerstones for us to design, prototype, and implement an infrastructure foundation for future SE bots.

In order to follow the above guidelines into action, this section also proceeds to derive a set of specific and actionable design requirements for managing and controlling various automation features. To satisfy most design guidelines above, the main idea behind prototyping a management console is to create an infrastructure console that orderly provides event subscription services, is compatible with various SE bots with control, and oversees bot and human activities. Its deployment has been decided on GitHub, the leading OSS code-hosting plat-

Table 4.1: Design guidelines of bot management systems

ID	Design Guideline	Empirical Evidence	Engineering Experience	Literature Evidence
DG1	Provide a series of stable and robust automation assistance with routine and repetitive tasks and support developers' daily workflow.	$\mathbf{RQ_{2.1}}$, $\mathbf{RQ_{3.1}}$	$\mathbf{RQ_{3.2}}$	[156, 180]
DG2	Enable customization for a butler bot, so its behaviors can accordingly fit the project and ecosystem norms.	$\mathbf{RQ_{2.3}}$, $\mathbf{RQ_{2.4}}$, $\mathbf{RQ_{3.1}}$	$\mathbf{RQ_{3.3}}$	-
DG3	View and control access permissions, and update dependencies in time for security.	$\mathbf{RQ_{3.1}}$	$\mathbf{RQ_{3.3}}$	[103, 180, 181]
DG4	Minimize identifiable information exposure in data pipelines for privacy concerns.	-	$\mathbf{RQ_{3.3}}$	-
DG5	Give simple and effective feedback during interactions with developers and provide controls over the types and frequencies of notifications.	-	$\mathbf{RQ_{3.2}}$, $\mathbf{RQ_{3.3}}$	[103, 181]
DG6	Promote automation component reuse and transparency.	$\mathbf{RQ_{3.1}}$	$\mathbf{RQ_{3.3}}$	[97, 103, 180]

form, and the native development environment for many existing SE bots [44, 103, 174, 180]. Especially, this prototype should support various mechanisms of bots relying on subscribing to different repository events, which groups data on commit, issue, pull-requests, and atomic repository events. Therefore, to derive these design guidelines into implementation specifications, this prototype requires the following requirements while implementing this prototype:

- Leverage a web framework for creating the interface of a management console, which employs a simple but effective UX design (**DG5**).
- Employ existing frameworks for powering bot-human interaction interface, whose framework is compatible with many prevalent automation features (**DG1, DG3**).
- Collect developer activity data at GitHub Event Timeline API and GHArchive data set with a controlled data pipeline. Also, integrate with the GitHub environment and subscribe to its webhook event system (**DG4, DG6**).
- Enable browsing and selecting automation functions from a catalog, and enable

customization for a repository *butler* (**DG2**).

- Control access permissions and notifications in a unified channel (**DG3**).

4.2.2 Architecture Decisions

Thus, supporting the implementation of a bot management console requires several "satellite" architectural components, as indicated in Figure 4.1. The high-level goal of this architecture is to remain compatible with most existing GitHub SE bots with rule-based design (see Chapter 3), and also provides support for future bots powered by AI and machine learning. Thus, its overall architecture follows an extensible event architecture [158, p. 556] with an event bus, GitHub Timeline, provided by GitHub [55, p. 55]. The bot management console is the main component coordinating other components of the overall design. Its user settings are saved in the `configuration storage`. An `event actor` is a unified agent that interacts with the GitHub event timeline for all automation features. A `history data collector` retrieves users' data from the GHArchive dataset and code repository for some bot features such as code reviewer recommendations. Finally, the `event subscriber` listens to the GitHub event timeline in real-time, particularly for some reactive SE bots' activities.

Bot Management Console

The console itself conducts four main functionalities. First of all, it helps developers to set up bots and automation features that they wish to deploy on their repositories. By connecting to external automation and bot catalog, developers are allowed to integrate bots from one particular entry. The second functionality is to manage authentication through a central portal. The management console is responsible to manage various bots' authentication in one secured portal by overseeing personal access tokens. Similarly, the repository access function should also distinguish access tokens for various repositories, and so that the console and its bots can operate on repositories with secured channels. Finally, to improve the transparency of bot activities, the console visualizes statistics of bots and human activities, including notifications, repository events, and automated flows [139]. Therefore, the prototype may serve as an enhancement of what the original "GitHub insights" provided, with a concentration on SE bots activities.

Configuration Storage

The configuration storage component has straightforward purposes as it stores three streams of data, permission settings, bot selection and configuration, and notification and communications settings. Permission settings include the level of permission given and restricted for each employed across various repositories. Bot selections and configurations are local settings for a single bot, which are often stored in the forms of `json` and `yml` according to the target bot's preference. Notification and communication settings are the major configurations of the notification control function provided by the bot management console.

Event Actor

The event actor is the only medium for posting various bot responses on GitHub's event timeline. This component collects automation services' responses from the target bot management console and merges all action requests through its secured data pipeline. By applying this single channel of post actions, the system avoids multiple event subscribers conflicting and overloading the code hosting platform. In addition, it can work with event timelines by sending repository updates in a batch. For instance, multiple bots may check a pull-request including the PR changeset, test execution, and deployment preview, etc. By leveraging an event actor, developers who subscribe to all repository events could only be notified once instead of several times.

History Data Collector

For many bots requiring historical event data, the History Data Collector helps to crawl data from longitudinal event databases such as GH Archive. For supporting the extensibility of future SE bots and providing more advanced functions such as AI and machine learning based recommendations, historical event data significantly enhances the precision of predictions and increases the accessibility to meaningful features [8, 34]. This component is designed to increase the extensibility and compatibility of future bots powered by AI and machine learning techniques.

Event Subscriber

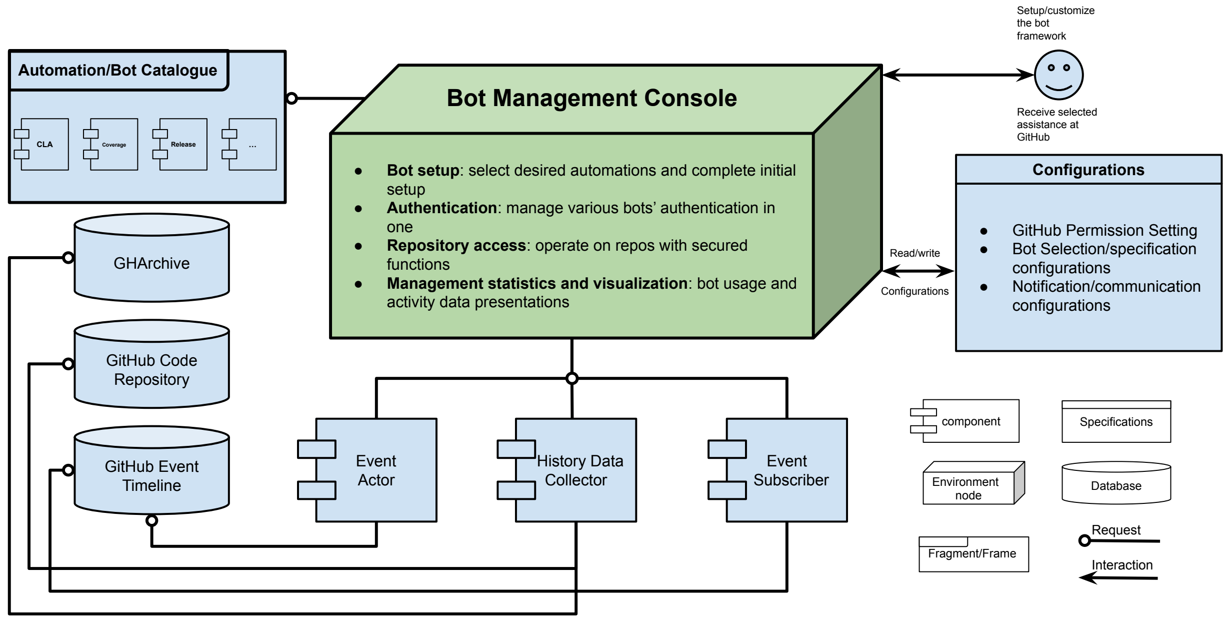
The event subscriber is a real-time event listener who monitors updates of repositories, and therefore other automation services may act their functionalities based on its data feed. Similar to the event actor, having a single event subscriber instead of multiple bots working in parallel aims to alleviate major risks of conflicts and data server overload.

External Data Sources

There are three major external data sources integrated with the bot management console. The first, and most substantial one is the GitHub Event Timeline[1]. This timeline handles major data IOs for SE bots and updates repository content. The second is the historical event database provided by GH Archive[2]. In addition to the analysis from Google BigQuery,

[1]Timeline events: `https://docs.github.com/en/rest/issues/timeline?apiVersion=2022-11-28`
[2]GH Archive: `https://www.gharchive.org/`

GH Archive provided a complete database of software engineering data within the platform of GitHub, which breaks the storing limit of GitHub timeline. Finally, the source code versioning control data from git systems complemented the above two data sources when there is a request for data feed at the source code level.



Setup/customize the bot framework
Receive selected assistance at GitHub
Automation/Bot Catalogue
CLA
Coverage
Release
...
Bot Management Console
Bot setup: select desired automations and complete initial setup
Authentication: manage various bots' authentication in one
Repository access: operate on repos with secured functions
Management statistics and visualization: bot usage and activity data presentations
Read/write
Configurations
Configurations
GitHub Permission Setting
Bot Selection/specification configurations
Notification/communication configurations
GHArchive
GitHub Code Repository
GitHub Event Timeline
Event Actor
History Data Collector
Event Subscriber
component
Specifications
Environment node
Database
Fragment/Frame
Request
Interaction

Figure 4.1: Architecture of proposed bot management console framework

4.3 Principle Use Cases

As I reported in Chapter 3, SE bots are becoming multi-tasking repository *butlers*. With the current practice, some “re-implementing the wheel” often happens when developers were crafting their customized housekeepers by assembling desired functionalities and selecting interaction mechanisms [174]. Therefore, the goal of the first principle use case is to create a pipeline for automating the bot creation and deployment process.

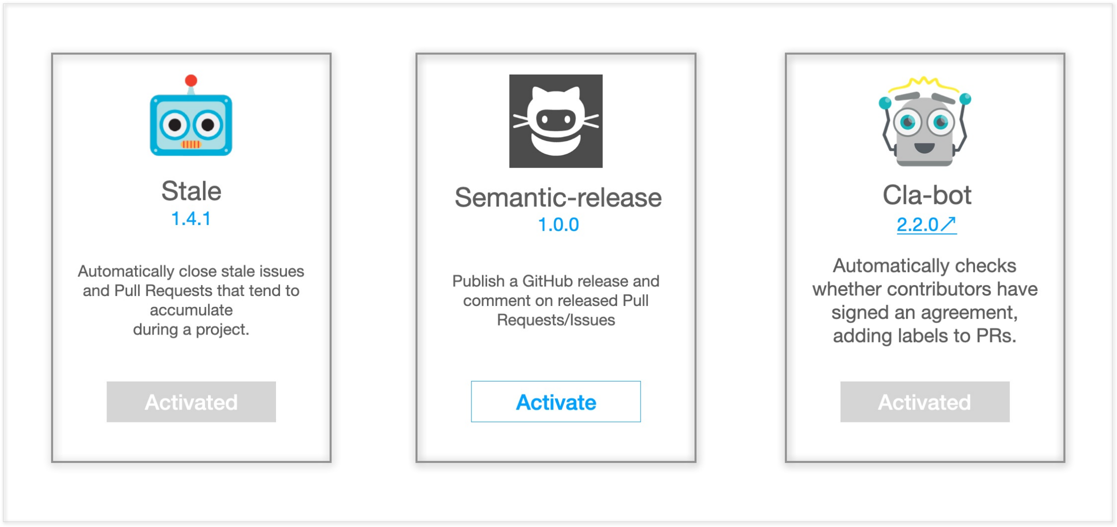

Stale
1.4.1
Automatically close stale issues and Pull Requests that tend to accumulate during a project.
Activated
Semantic-release
1.0.0
Publish a GitHub release and comment on released Pull Requests/Issues
Activate
Cla-bot
2.2.0↗
Automatically checks whether contributors have signed an agreement, adding labels to PRs.
Activated

Figure 4.2: User Interface for automation selection

Automation Browser

The user activates the console application, either by clicking on the application icon in the GitHub interface or its web application portal. Upon opening, the first tab that the user sees is the Automation Browse. This tab should be designed with clear labeling and intuitive navigation.

Overview of Automation Browser: The Automation Browser is a library or repository of various software engineering bots (see Figure 4.2). These bots are automated tools or scripts that perform specific tasks related to software development, such as code testing, bug tracking, performance monitoring, and more.

Browsing Available Bots: The user can scroll through the list of available bots. Each bot is accompanied by a brief description of its functions, typical use cases, and any requirements or dependencies.

Bot Details: By clicking on a bot, the user can see more detailed information, including comprehensive documentation, user reviews, and a link to the bot's source code or project page. This allows users to make informed decisions about whether the bot will be useful for

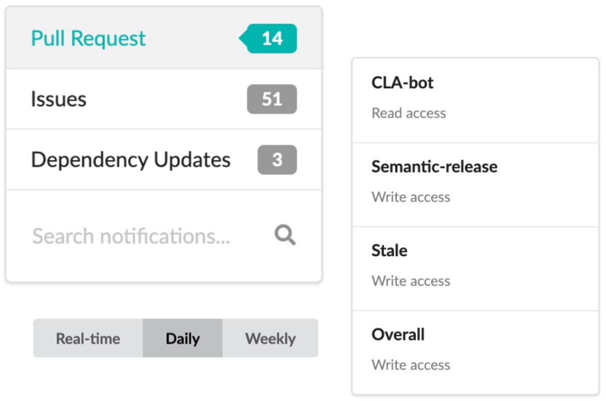


Figure 4.3: UI for controlling notification and access permission

their specific needs.

Installing and Configuring Bots: If the user decides to use a bot, they can click on an "Install" button. This will guide the user through the installation process, including any necessary configuration settings. Some bots may also offer a "Test Run" option, allowing users to see the bot in action before fully installing it.

Support and Updates: The Automation Browser also provides support resources for each bot, such as tutorials, FAQs, and contact information for the bot's developers. Additionally, the Automation Browser keeps track of updates to the bots, notifying users when an update is available.

Closing the Automation Browser: Once the user is finished exploring the Automation Browser, they can close the tab or navigate to other sections of the console application. Their selections and settings in the Automation Browser will be saved for future sessions.

Control Notification and Permission

Accessing the Console: The user, typically a project manager or developer, opens the bot management console. This could be via a web interface, a desktop application, or a command-line tool, depending on the specific implementation.

Navigating to Bot Management: The user navigates to the section of the console dedicated to bot management. This could be labeled ”Bot Management,” ”Automation,” or similar.

Selecting a Bot: The user selects a bot from the list of bots currently installed and being used in their software project. Each bot should have a clear name and brief description to help the user make their selection.

Accessing Notification Settings: Once a bot is selected, the user can access the bot’s settings. Among these settings, there should be a section specifically for managing notifications. The user navigates to this “Notification Settings” section.

Customizing Notifications: In the Notification Settings, the user can customize when and how they receive notifications from the bot. For example, they might choose to receive an email whenever the bot identifies a new bug, or they might configure the bot to only send notifications for high-priority events. The user can also choose which team members receive notifications, depending on their roles and responsibilities.

Managing Bot Permissions: In addition to managing notifications, the user can also control the bot’s permissions in this settings section. For example, they might allow the bot to create new issues in their project management tool, or restrict it to only reading and commenting on existing issues. They could also control which parts of their codebase the bot has access to.

Saving Settings: After making changes to the notification settings or bot permissions, the user saves their changes. The console should confirm that the changes have been saved, and the new settings should take effect immediately.

Returning to Bot Management: Once they are finished managing the bot’s notifications and permissions, the user can return to the main bot management page, close the console, or navigate to another section of the console.

Repository Activity

Opening the Console: The user, likely a project manager or software developer, opens the bot management console. The console could be accessed through a web interface, a desktop application, or even a command-line tool, depending on the specific implementation.

Navigating to the Repository Monitoring Section: The user navigates to the section of the console designed for repository monitoring.

Selecting a Repository: The user selects a repository from a list of repositories that are currently being monitored. Each repository should have a clear name and a brief description to help guide the user's selection.

Viewing Activity Feed: Once a repository is selected, the user is presented with an activity feed. This feed displays recent actions and changes in the repository, such as new commits, pull requests, issues, and comments. The feed is typically organized in reverse chronological order, with the most recent activity at the top.

Filtering and Searching Activity: The user can filter or search the activity feed to find specific events. Filters might include date ranges, types of activity (e.g., commits, pull requests), or specific users. A search bar can also help the user find activity related to specific keywords or phrases.

Interacting with Activity: Depending on the console's capabilities, the user might be able to interact with the activity directly from the feed. For example, they could click on a commit to view its details, respond to a comment, or merge a pull request.

Exiting the Repository Monitoring Section: After they've finished monitoring the repository activity, the user can navigate to another section of the console, return to the main page, or close the console. Any filters or notification settings should be saved for the next time the user accesses the repository monitoring section.

The design guidelines, architectural decisions, and principal use cases presented in the above sections directly address the posed $\mathbf{RQ_{4.1}}$. The meticulously crafted design guidelines provide a detailed roadmap, steering the conceptualization and development of solutions to our research problem. Concurrently, the architectural decisions, informed by rigorous empirical studies and theoretical development, ensure the robustness and adaptability of our solutions, considering both current requirements and potential future expansions. The principal use cases, on the other hand, serve as practical illustrations of how these solutions operate in real-world scenarios. They not only validate the effectiveness of our proposed solutions in context but also elucidate their potential impact and value for end-users. Collectively, we may answer this RQ with the following:

Following the design guidelines based upon empirical results and theories in the literature, a management console for software engineering automation can leverage an event-subscriber style of architecture, which supports major use cases including automation browser, notification and permission summary, and activity visualization.

4.4 Evaluation though Simulated Bot Deployment

Employing SE bots in real-time OSS production or experimenting with OSS maintaining teams can be exceptionally expensive [50, 90]. Therefore, we take an approach of agent-based simulation to study this complex OSS software environment which is composed of multiple agents, including various levels of contributors and SE bots [23, 117, 149]. This simulation study is advised by ODD (Overview, Design concepts, and Details) protocol, which summarizes how the simulation will be conducted for this agent-based models [69]. The overall research questions of this evaluation study are:

RQ$_{4.2}$ *How do different combinations of SE bots influence repository activities and notification responses in an OSS project?*

Due to the event-based triggering mechanism, one of the primary concerns of bot usability is that bots disruptively abuse notifications of a repository [103, 180]. This effect usually escalates when the repository has employed multiple bots or receives increasing external contributions. Since there was no perfect real-world example of adopting several target bots on one single project, this part of the evaluation is based on empirical data and a rule-based simulation to compare project-generated notifications with or without a control mechanism. Therefore, the second research question of this simulation study is,

RQ$_{4.3}$ *What is the effectiveness of various notification management strategies in mitigating interruptions among elite developers?*

This evaluation leverages combinations of three open-source bots and simulates how they would produce notifications (see Figure 4.4 for a high-level overview of the model). In the first phase, I extract the atomic operating mechanism of each bot. For example, based on the default setting of `Stale`, this atomic bot would label an Issue after 60 days of inactivity, if the issue was not a pinned or security Issue, and also close the Issue for another seven days without activity. This study extracts these reactive rules that how bots work with repository events based on their shared source code. In the second phase, we collect target repositories' event data, and based on simulated notification trends based on their reactive rules, simulate how many bot-generated events would be. A summary for calculating each type of repository event and further notifications are illustrated below.

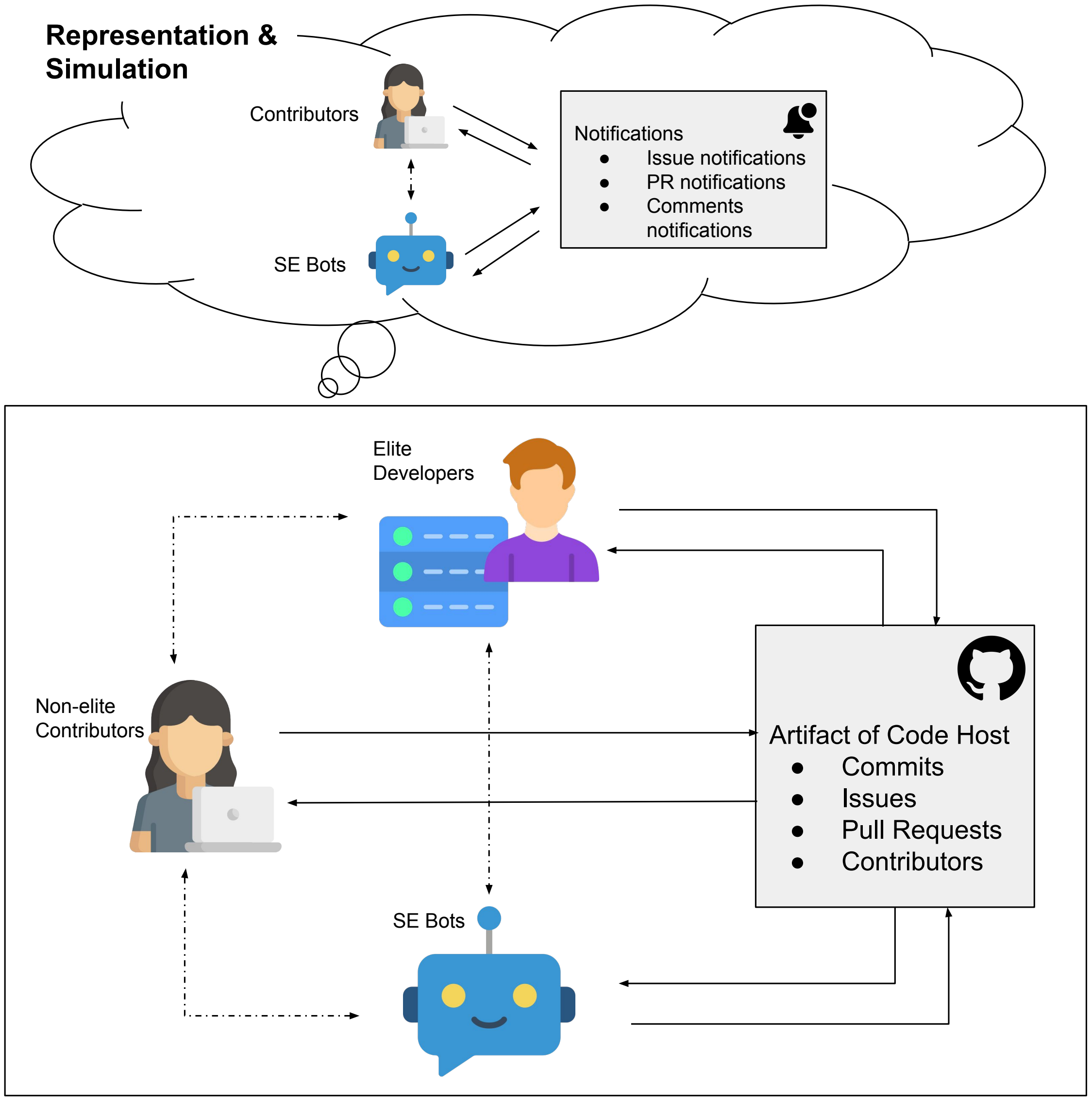


Figure 4.4: The agent-based system described in this simulation study. The lower part of the figure presents the real-world interactions between actors and the code host environment, and the upper part is the representation and simulation of notifications produced by this agent-based model. In addition, solid lines represent interactions with the environment, and the dashed lines represent communications between agents.

4.4.1 ODD Protocol

ODD (Overview, Design concepts, and Details) protocol is a standardized method to describe individual-based and agent-based models, especially in the field of ecological modeling [68, 69]. Its structured approach ensures that every important aspect of the model is comprehensively and consistently described, enhancing the study's transparency and interpretability. Utilizing the ODD protocol facilitates the understanding of the model by the research team themselves and other software engineering researchers, which is critical to the validation, replication, and extension of the model. Furthermore, by standardizing the description of agent-based models, the ODD protocol allows for more effective comparison and synthesis of different models, promoting knowledge accumulation and theoretical development in the field: the simulation application in Software Engineering. Particularly in this study, we follow Grimm et al.'s 2010 updated definition of ODD protocol to describe all essential elements for conducting bot deployment simulation. The following of this section describe the essential elements of this model, and Figure 4.4 provides an overview of the model representation.

Purpose

The main purpose of this simulation is to evaluate the performance of combinations of SE bots in terms of interrupts and provide useful insights into how the bot might perform in different scenarios. The use of empirical data from developer activities and rule-based implementation can also add credibility to the simulation results, as it helps ensure that the simulation is grounded in real-world data.

Entities, State Variables, and Scales

Similar to other ecological models, the main entities of this model include agents/individuals, spatial units, environments, and collectives.

Agents/individuals: main agents of this model include developers and bots. Particularly, developers include non-elite developers who do not have administrative privileges and elite developers who perform all sorts of maintaining activities (see Collectives later). SE bots were major ones described in earlier sections of Chapter 3.

Spatial units: since the majority of the observation and simulation happen online, the spatial units are not limited to the physical space but should be categorized as repositories. In this model, we leverage the six repositories of the large Data Science framework/package of Python. The target repositories include five Python Data Science projects, including, Keras, Matplotlib, nltk, numpy and pandas, the most active repositories/projects from Chapter 2. These repositories share a similar development environment supported by Python packages, and therefore align with each other in terms of development practice. Particularly, selecting these sample repositories from the same ecosystem for this simulation study is crucial to maintaining consistency and comparability in the analysis. The characteristics and behaviors of repositories can vary significantly across different ecosystems due to differences in language conventions, development practices, community norms, and platform features. By focusing on repositories within the same ecosystem, we control for these ecosystem-level factors, allowing us to more accurately isolate the effects of the variables we're interested in studying. This also enhances the external validity of our simulation study, as the findings will be more generalizable to other repositories within the same ecosystem. Lastly, using repositories from the same ecosystem simplifies the data collection and analysis processes, as the repositories will have similar structures and contributing practice. Additionally, some developer agents have contributed activities across multiple projects.

Environment: the target environment of this model is the GitHub code hosting environment, where artifacts and contributing activities are open to public users.

Collectives: there are two main collectives of each repository's developer community: elite developers and non-elite community contributors. Elite developers are the ones who have full write access to the repository, so they are able to perform all major repository maintenance activities. Non-elite community contributors may contribute by following the repository's contributing guidelines, but their work requires an extensive evaluation of the internal team.

Process Overview and Scheduling

The overall process follows the description of the GitHub event system and developers' reported workflow in Chapter 3. Elite and non-elite developers' behaviors were observed based on the empirical GitHub event data from January 1st, 2022, to December 30th, 2022. Depending on which type of bot it was, the produced notification follows a *proactive* or *reactive* workflow (see Section 3.2.2).

Besides developer workflow, the main three SE bots agents deployed in this system are the following:

- **Stale**: `stale`[3] is an issue management bot that reminds developers to take action towards inactive issues and pull requests, and therefore maintain an organized issue repository. By analyzing its documentation, (default settings) the stale bot's behavior process and schedule can be summarized as the following pseudocode in Algorithm 1.
- **Welcome**: `welcome`[4] (or `new-issue-welcome`) is a simple community management bot that welcomes and acknowledges first-time contributors to a repository. `welcome` helps post a welcome message with required information about contributing to this

[3]probot/stale: `https://github.com/probot/stale`

[4]welcome: `https://github.com/behaviorbot/new-issue-welcome`

Algorithm 1 Stale Bot Behavior

```
procedure ProcessIssuesAndPRs
    for each Issue and PR in the Repository do
        if isLabeled(Issue or PR, staleLabel) then
            if isClosed(Issue or PR) or isLocked(Issue or PR) then
                continue
            end if
            if hasRecentInteraction(Issue or PR) then
                removeLabel(Issue or PR, staleLabel)
            else
                close(Issue or PR)
            end if
        else
            if isStale(Issue or PR) and not isExempt(Issue or PR) then
                addLabel(Issue or PR, staleLabel)
                postComment(Issue or PR, staleMessage)
            end if
        end if
    end for
end procedure
```

repository, and its behaviors can be summarized with the following pseudo-code in Algorithm 2.

Algorithm 2 Welcome Bot Behavior

```
procedure WelcomeNewContributors
    for each NewEvent in the Repository do
        if isNewUser(NewEvent.user) then
            if isIssue(NewEvent) then
                postComment(NewEvent, welcomeMessageIssue)
            else if isPullRequest(NewEvent) then
                postComment(NewEvent, welcomeMessagePR)
            end if
        end if
    end for
end procedure
```

- **Coveralls**: `coveralls`[5] is a set of functions that help report the test coverage of the project. This service has a variety of support in its back-end with various program-

[5]coveralls: `https://coveralls.io/`

ming languages and test execution mechanisms. However, its front-end services can be described in the following pseudo-code in Algorithm 3.

Algorithm 3 Coveralls Bot Behavior

```
1: procedure ProcessCoverageReport
2:     for each NewCommit in the Repository do
3:         report = runCoverageCheck(NewCommit)
4:         if isNewPullRequest(NewCommit) then
5:             postCoverageComment(NewCommit, report)
6:         end if
7:         if coverageDecreased(report) then
8:             postCoverageAlert(NewCommit, report)
9:         end if
10:    end for
11: end procedure
```

For the simplicity of simulation model design, we only adopt and keep the default settings for these bots, e.g., the expiration time of an issue or pull request remains unchanged over the entire simulation period.

Design Concepts

This model has made the following major assumptions when simulating notifications of various combinations of bots. First, activities performed by humans would not change significantly after the bot adoption, in terms of adjusting behaviors, i.e., we can still rely on existing empirical data as an observation of software engineering activities. Second, this evaluation assumes that each developer employs the default settings of getting GitHub notifications[6], i.e., the developer receives notifications about any events on *watched* repository, teams, and conversations[7]. We may estimate the overall notification intensity based on repository events since we cannot replicate a precise number of notifications received by each developer. Fi-

[6] GitHub notification configuration: `https://tinyurl.com/yc43cbmh`

[7] *Conversation* on GitHub refers to comments on Issues and Pull Requests that the developer is participating in or watching, Pull Request reviews, and Pull Request pushes.

nally, for an atomic bot, this evaluation assumes bot adoption only associates with particular activities as an intervention, and they remain default settings without customization, e.g., an Issue-related bot would only influence Issue events.

Particularly, for a repository *repo*, there are n issues, m pull requests, and l commits created within a given period of time. Let us denote arbitrary issue as i, pull request as j, and commit as k. Assuming elite developers of this *repo* subscribe to all repository notifications according to the default setting, the number of all generated notifications N_{repo} equals the sum of notifications from issues, pull requests, and comments, i.e.,

$$N_{repo} = N_{issue} + N_{pullrequest} + N_{comment} \tag{4.1}$$

and specifically, notifications from issues and pull requests are mainly based on their creation and events that happened,

$$\begin{aligned} N_{issue} &= \sum_{i=1}^{n} Event_i + n \\ N_{pullrequest} &= \sum_{j=1}^{m} Event_j + m \\ N_{comment} &= \sum_{i=1}^{n} Comment_i + \sum_{j=1}^{m} Comment_j + \sum_{k=1}^{l} Comment_k \end{aligned} \tag{4.2}$$

Notification Summary Mechanism

The "Notification Summary" approach is designed to manage and streamline bot notifications, thereby reducing disruptions to developers. This approach operates on the principle of aggregation and time-bound delivery, aiming to mitigate the impact of non-urgent notifications on a developer's workflow.

In this method, instead of delivering notifications as soon as they are triggered, the system collects and stores non-urgent bot notifications over a specified period of time. These could be notifications related to minor updates, non-critical system information, or changes that do not require immediate attention or action from the developers.

Once the pre-determined period elapses, the system generates a summary of all collected notifications and sends it to the developers. This could be arranged in a digest format, categorizing notifications based on their types, priority levels, or associated tasks, to facilitate easy review and action from the developer's side.

By consolidating multiple notifications into a single summary report, this approach substantially reduces the frequency of interruptions, allowing developers to focus more on their primary tasks. Moreover, by providing a comprehensive overview of all non-urgent notifications at once, it allows developers to understand the broader context and make more informed decisions about their subsequent tasks.

Overall, the Notification Summary approach represents a significant advancement in bot notification management, balancing the need for developers to stay informed with the necessity of maintaining an efficient and interruption-free development environment.

4.4.2 Analysis Methods

The final phase of this evaluation leverages hypothesis tests with one-way ANOVA to compare the number of notifications with different bot combinations and notifications control settings.

This simulation study intends to evaluate whether notification control settings may effectively reduce the number of notifications when a repository adopts a combination of bots. Therefore, it tests whether this management console may improve elite and expert developers' experience with bot notifications.

4.4.3 Evaluation Results

In this evaluation study, we utilize an agent-based model to simulate the complex dynamics of five software development projects. Our simulation environment is populated by autonomous agents, each representing a developer or a software bot, who interact with each other and with the project's artifact such as the codebase. These agents have diverse behaviors mirroring the heterogeneity observed in real-world development teams, i.e., empirical OSS software engineering data. By running the simulation under various conditions and parameters, i.e., the number and combination of SE deployments, we can investigate how these factors influence project notifications. This agent-based approach offers a powerful tool for understanding and managing the intricate, dynamic processes involved in OSS development.

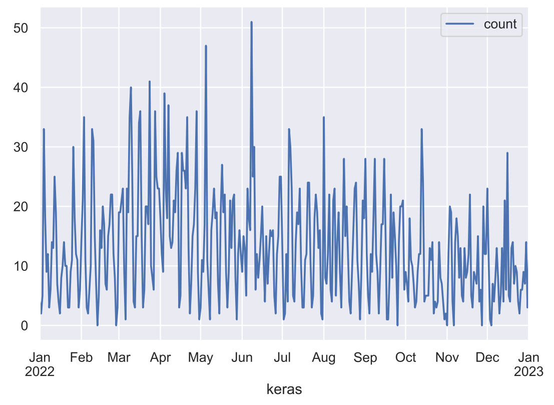
We first conducted an Augmented Dickey-Fuller test to test for stationarity in our time series data. The test statistic was -2.86, with 1 lag used based on the Schwarz Information Criterion, and 1825 total observations (five projects in one year). The critical values for the test statistic at the 1%, 5%, and 10% levels are -3.50, -2.89, and -2.58, respectively. Because the test statistic is more negative than the critical value at the 5% level, we can reject the null hypothesis of the presence of a unit root at the 5% level. Thus, we conclude that our time series is stationary.

We conducted a one-way ANOVA to compare the effect of bot deployment on three different groups: control, sending by hourly batch, and sending by daily batch. The results showed a significant effect of treatment on the outcome, $F_{(4,1820)} = 5.43$, $p = .01$. Post hoc comparisons using the Tukey HSD test indicated that the mean score for group control (M = 5.2, SD = 1.3) was significantly different than group daily batch (M = 3.1, SD = 0.9). However, the group daily batch (M = 4.7, SD = 1.1) did not significantly differ from the group hourly batch.

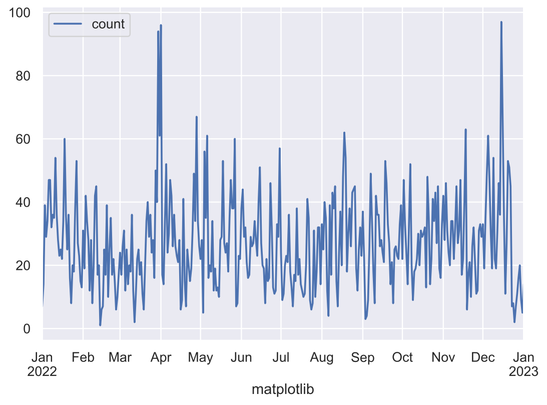
In this section, we describe developer activities and SE bot's reactions with an ODD protocol to build an agent-based model, and therefore, we could simulate several bot activities based on empirical data. By comparing the number of simulated notifications, we can answer $\mathbf{RQ_{4.2}}$ and $\mathbf{RQ_{4.3}}$ with the following:

Answers to $\mathbf{RQ_{4.2}}$: In addressing the research question, we employed an agent-based model to simulate the impact of deploying SE bots on Python data science projects. Our findings indicate that the introduction of these bots significantly amplified the repository activities executed autonomously. As a consequence, the number of repository notifications for its watchers, such as elite developers, could experiencee a marked increase.

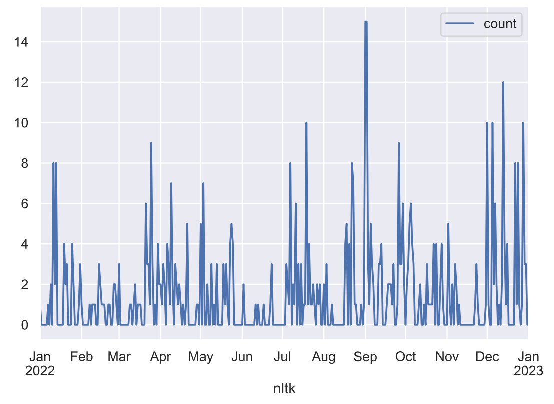
Answers to $\mathbf{RQ_{4.3}}$: Based on the one-way ANOVA test, we were able to reject the null hypothesis. Consequently, a significant difference was identified between scenarios with and without notification controls, highlighting the impact of these controls provided by the management console. However, when considering the varying lengths for notification batches, our data did not reveal any significant differences, as evidenced by the results of the Tukey HSD Test. This suggests that the duration of notification batches may not be a substantial factor influencing the observed phenomena.



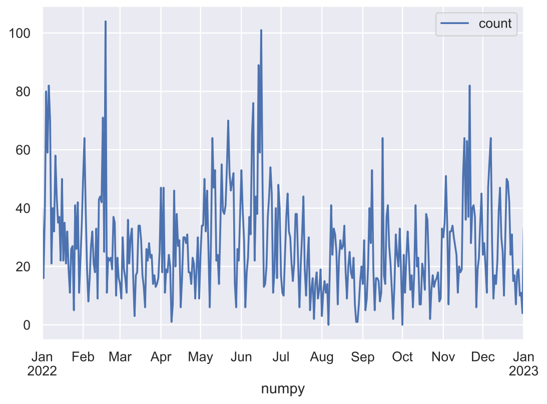
(a)



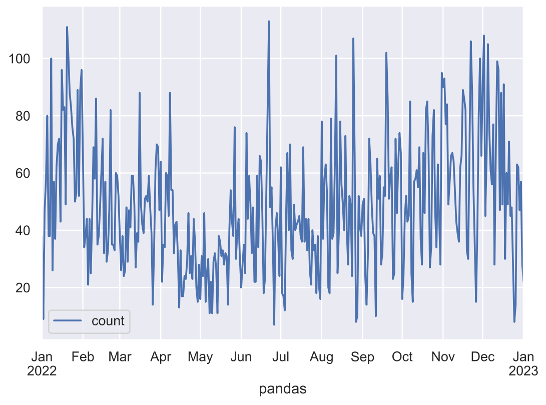
(b)



(c)

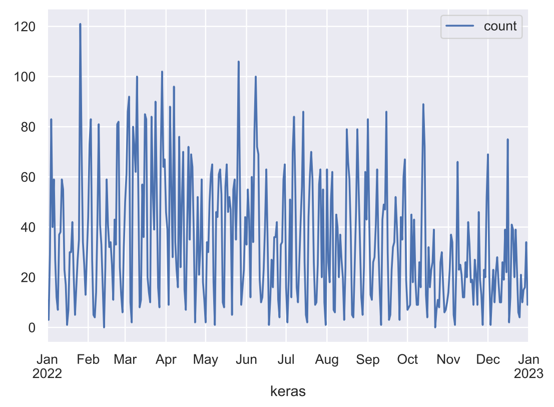


(d)

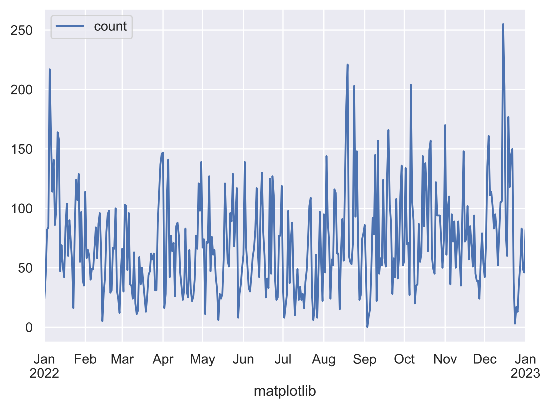


(e)

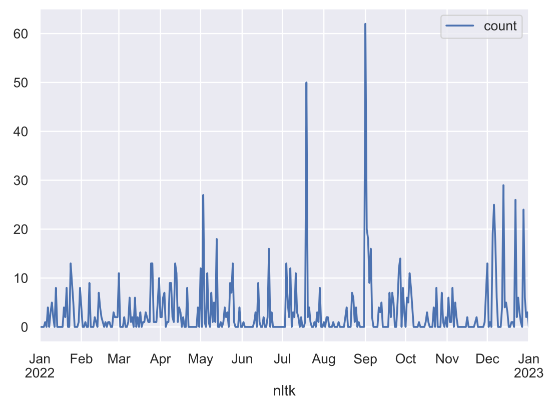
Figure 4.5: Notifications generated from comments for five sampled repositories



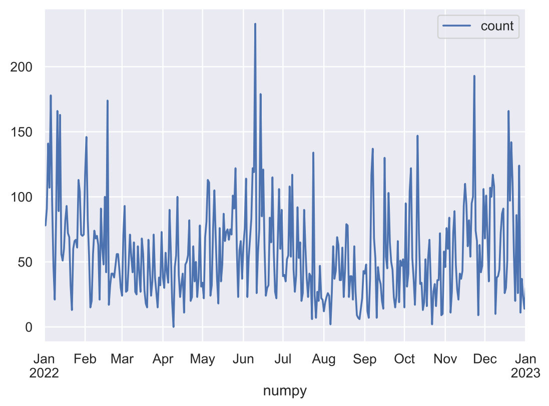
(a)



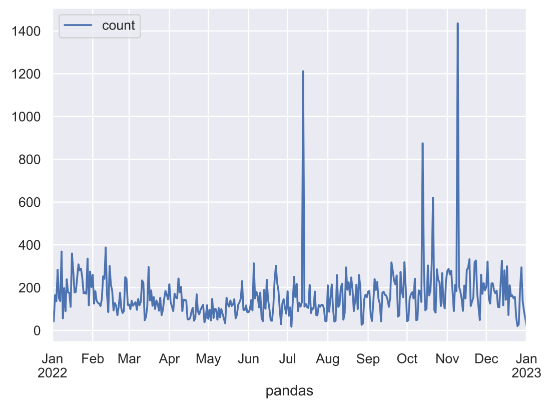
(b)



(c)

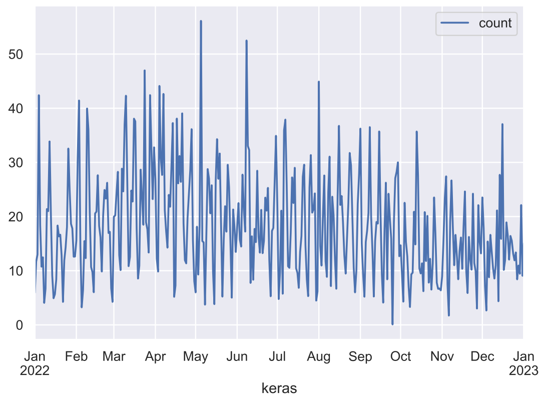


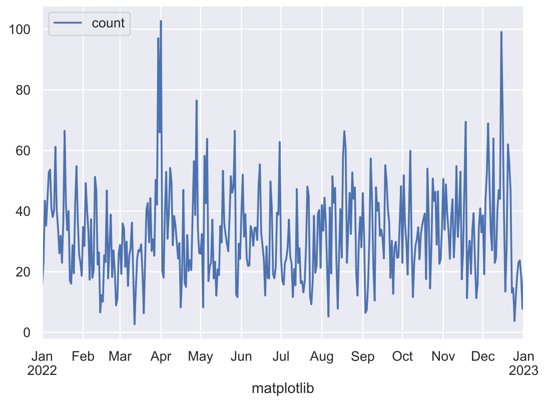
(d)

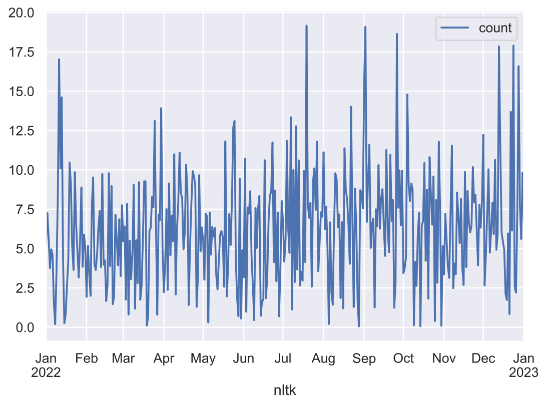


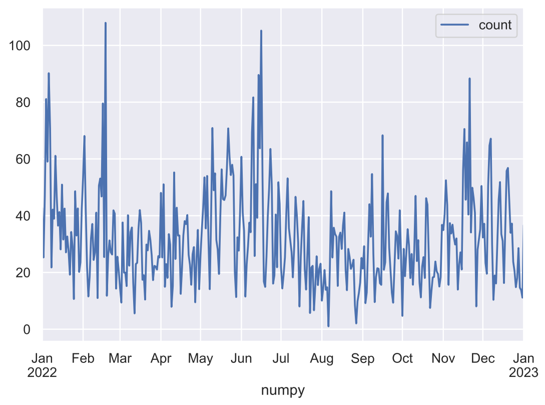
(e)

Figure 4.6: Notifications generated from issue and pull request for five sampled repositories


(a)


(b)


(c)


(d)

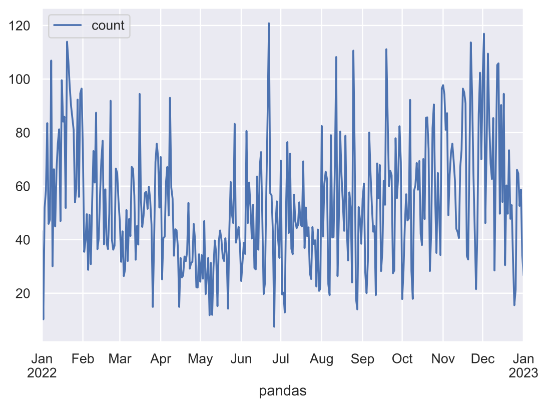

(e)

Figure 4.7: Repository notification from comments with three combinations of bot deployed

Chapter 5

Discussion

5.1 Empirical Studies on Elite Developers

This section discusses the findings of the previous two empirical studies with elite developers in OSS communities. Based on analysis methods of research questions, this section first presents in-depth discussions of descriptive ones including $\mathbf{RQ_{2.1}}$, $\mathbf{RQ_{2.2}}$, and $\mathbf{RQ_{2.4}}$, and second, inferential ones including $\mathbf{RQ_{2.3}}$ and $\mathbf{RQ_{2.5}}$. Finally, this section presents their research and design implications.

5.1.1 Contribution Activity and Trend of Elite Developers

First of all, my findings on elite developers' activities confirm their crucial roles in OSS development. As the result of $\mathbf{RQ_{2.1}}$ shows, elite developers have engaged in the majority of the projects' activities, although they only account for a small proportion of contributors in the larger community. Except for communicative activities, elite developers contributed to over 50% of activities in all the other three categories. The results confirm prior literature dating back to the early 2000s [39, 119, 189]. We can conclude that OSS projects are still largely driven by a small number of elite and expert members even after over 20 years of evolution. While high concentrations may ensure bottom-line project outcomes, I argue that such a situation is not optimal for the long-term health and sustainably of an OSS project [38]. Engaging the non-elite users' participation through mechanism and technology innovation has remained a critical challenge [152]. To sum up, the results of $\mathbf{RQ_{2.1}}$ confirm and extend the findings in the prior study focused on developers' workload [189]. From the role-specific perspective (elite vs. non-elite), this work reveals that the imbalance of workload does not only exist in technical contributions but also happens in non-technical activities, where situations can be even worse.

Secondly, the results of $\mathbf{RQ_{2.2}}$ show that in most of the sampled projects elite developers performed different activities as projects grew. 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 [10, 121]. For example, a developer, while initiating the project, was heavily involved in contributions to source code artifacts. By contributing to these technical activities, they intend to build their technical competencies and establish their community reputation. However, as the project and community grew, they found themselves having to shift their focus to supporting and communicating with novice contributors and users. Moreover, imposing additional burdens on management since the project community expanded is often less effective without necessary training and self-motivation. Although developers vary in their personal perceptions about how they would develop their competencies [93], technical reputations and development may not always be the solo pursuit of having an OSS-related career. However, at least in the software engineering research community to date (except reflected in a few very recent studies, e.g. in [159]), this issue has not received sufficient attention. Future research is necessary to thoroughly investigate and address the issues related to role transitions in OSS communities, especially unintended ones.

Finally, $\mathbf{RQ_{2.4}}$'s findings convey another essential message about elite developers that there are substantial differences among ecosystems regarding the proportion of activities done by the elite developers. This message suggests that the elites behave in different patterns between various ecosystems, and I argue that their activity profiles are associated with the nature of the ecosystem. One example is the *Python Data Science* ecosystem. In this ecosystem, elite developers spend most of their efforts in communicative and supportive activities (see Table 2.8). This fits its nature as a multidisciplinary ecosystem, which consists of mathematicians, machine learning researchers, and software engineers [127]. Even effectively organizing and supporting their collaboration could cost much effort according to the team science literature [56, 122]. Moreover, this **RQ**'s results also suggest that elite developers from the *Firefox Add-ons* ecosystem account for a significantly higher proportion of all activity categories. Given that most add-on projects are small and easy ones, without a large contributor base, they often have a hard time attracting and retaining other contributors from their community.

5.1.2 Impact on OSS Communities and Artifacts

The findings of $\mathbf{RQ_{2.3}}$ reveal relationships between elite developers' effort distributions and project outcomes. In general, there are negative associations between non-coding activities and technical project outcomes. For three out of four project outcome indicators ($NewC_{im}$, BCT_{im}, and $NewB_{im}$), these results suggest that the trend of increasing efforts into communicative, organizational, and supportive work is negatively correlated with the project outcomes. Here I argue that the negative associations were due to the increasing involvement of non-elite community contributors and the focus shifts of elite developers. Typically, as a project's community grows, external non-elite developers may contribute to the project in various ways, including suggesting features, reporting bugs, or submitting their own patches [39, 82]. Since non-elite developers often do not have a comparable level of technical expertise or familiarity with the project code base, their code could be more buggy, thus may lead to lower software quality [2]. 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 elite's efforts in supportive work have positive correlations with project quality. My 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$_{\mathbf{2.3}}$'s findings, if put together, describe a dilemma that elite developers gradually have to face in OSS communities: With the growth of their projects and contributor community, they have to spend more time on non-technical tasks, which forces them to reduce their efforts on technical contributions or devote more overall time into maintaining their OSS projects. Since their technical activities still account for a majority of the project's typical development work (see Table 2.3), the project could experience productivity and quality loss. Yet, non-technical work has its own value by helping maintain expected practices of a project's engineering processes (e.g., defect removal process) and pay off with quality gains.

Another finding worth noting is the difference between non-company-sponsored and company-sponsored projects. Particularly, **RQ$_{\mathbf{2.3}}$**'s results indicate that company-sponsored projects tend to be more influenced by their elite developers' effort distributions. This is not surprising since maintaining these projects often relies on a small number of full-time employees acting as their elite developers, and this finding also empirically confirms other early qualitative studies, e.g., [169]. Findings of this study support that the involvement of industrial companies including economic incentives, compensations, supports, and potential career development opportunities could have led to higher motivations of community contributors in these company-sponsored projects [99, 111]. Moreover, GitHub recently provides a "sponsors" feature to financially support OSS developers; we may expect increased influence on conventional OSS projects, which are largely based on voluntary contribution [99].

The findings of $\mathbf{RQ_{2.5}}$ highlight two major points for discussion. First, for most sampled ecosystems, the trend of elite developers' activity can also predict project outcomes quantitatively at the ecosystem level. Findings of target ecosystems suggest that there are significant associations between ecosystem-wide productivity loss and increasing efforts of elite developers in non-coding activity (including communicative and supportive). Second, regarding project quality, an observed pattern among ecosystems is that efforts in organizational activities are positively associated with identified bugs, i.e., negatively associated with project quality.

Regarding ecosystem-level productivity, the relationships are consistent with slight variations, with the exception of the *Firefox Add-ons* ecosystem. In particular, the effects are much stronger for the *Amazon AWS* ecosystem (medium), followed by *npm/Node.js* and *Python Data Science* (small), then by *Eclipse* (barely small), and finally *Firefox Add-ons* (not significant). I argue that this phenomenon results from technical platforms' barrier and contributor base of their programming language among these ecosystems. Most of *Python Data Science* projects are written in Python. Even for ones that have a considerable portion of C/C++ files, e.g., `SciPy` and `NumPy`, Python still accounts for over 50% of its code base. All *npm/Node.js* projects are mostly written in JavaScript. Many studies have consistently concluded that Python and JavaScript are two accessible programming languages for novice developers to learn and excel with compared to C/C++ and Java [91, 110, 123, 148]. When elite developers were unavailable or distracted from devoting themselves to typical coding tasks, a larger base of community contributors could fill this gap without concerns about technical obstacles introduced by the platform's main programming language. In contrast, most *AWS* projects are written in C/C++, and thus these less popular programming languages resulted in a smaller community of OSS contributor base, and a hard time ramping up with technical tasks while worrying about the language's low-level technicalities and optimizations. Furthermore, most AWS projects focus on system-level programming for ultra-large distributed systems rather than application-level programming, sometimes even requiring certain levels of hardware knowledge. Thus, its own elite developers' effort allocations have a more significant effect in the *AWS* ecosystem. Finally, developers in the *Eclipse* ecosystem are more homogeneous, most being professional software developers, as a major difference compared to many other OSS ecosystems [13]. Thus, the knowledge gap in technical expertise between elite developers and non-elite community contributors assume to be smaller than in other ecosystems.

Regarding ecosystem-level quality, the predictive relationships are diverse and less significant. However, a notable exception is the *Python Data Science* ecosystem, in which the elites' efforts in organizational activities exhibit a strong effect size on quality metrics. I argue that this observation resulted from interdisciplinary team formation as mentioned before, i.e., teams consisting of developers, mathematicians, machine learning researchers, etc. Thus, coordinating and supporting its developer community, for instance assigning code review tasks to ones with adequate expertise, become critical and challenging. Thus, while performing these related organizational activities, elite developers fulfill their roles as *knowledge brokers* in these interdisciplinary teams [5, 58, 75].

5.1.3 Practical Implications

These two studies found and confirmed that imbalances of workload are prevalent in OSS communities, and have led to heavy burdens on a few central individuals such as elite developers [189]. First, in addition to the unbalanced technical workload identified in prior studies, my studies further found that situations are even worse for non-technical workloads (communicative, organizational, and supportive). The imbalances of non-technical workload can result in a huge order of magnitude (see Section 2.1.2 and 2.2.2). By examining the projects in these samples, I found the expansion of the elite developer group has failed to keep up with the pace of project and community growth, which cannot be spontaneously fixed in the evolution of projects. The slow expansion of the elite developer group reflects a conservative approach when guaranteeing members permission to perform administrative tasks. While OSS ideology is fairly progressive, its management structure is pre-industrial, i.e., only a few central individuals share of the authority and power in the community [33, 146, 168]. Improving community governance by decentralizing such authority and power, particularly related to routine work and repetitive tasks, could be a possible advance. This possible advance would not only alleviate elite developers' heavy burdens but also provide extra incentives for others in communities to contribute [136]. Moreover, allowing non-elite community contributors to share elite developers' routine duties would help offset the negative impacts of core member turnover [162]. In fact, decentralizing and delegating proportions of elite developers' work has already been recognized by practitioners. Recently, GitHub acknowledged that inadequate governance and excessive workload of a few core individuals are two major threats to project sustainability[1]. They also mentioned that allowing and guiding ordinary members to run the project is critical to address these threats. Therefore, this advance can be expected in the near future, particularly since platforms like GitHub have supported and committed to providing facilities for helping projects to implement such decentralization and delegation.

Second, $\mathbf{RQ_{2.3}}$'s findings also highlight that relationships between project outcomes and elite developers' efforts in non-technical tasks are more significant for company-sponsored projects. As opposed to non-company-sponsored projects, company-sponsored projects often inherit the management practices of the parent corporation, and their elite developers tend to be trapped more in routine non-technical work. In Wagstrom's dissertation, he has shown

[1]GitHub development blog, *Let's talk about open source sustainability* `https://github.blog/2019-01-17-lets-talk-about-open-source-sustainability/`

that the vertical integration between companies and OSS communities would inevitably lead to increases in unnecessary communicative and organizational practices [169]. Given the limited time and attention resources of developers, their unnecessary non-technical tasks could have hurt their projects' productivity. Thus, he recommended focusing on necessary communication only "meeting individual coordination requirements." According to the results of this chapter, his recommendation is still valid. From a company's perspective, avoiding directly "copying" their internal governing structures is necessary, even for the projects dominated by these companies [63, 143, 178].

5.1.4 Design Implications

With the growth of the project, elite developers often have to put more effort into communicative and supportive tasks. Our study reveals such a shifting of work may have negative impacts on project outcomes. As we present in the results section, these tasks are often necessary and cannot be ignored. Moreover, 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. The main time cost for them is to identify the external assignee. These tasks can be easily automated with tools [8]. 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 [31]. Automatic subscribe and unsubscribe can be realized through learning users' characteristics [19]. For documentation tasks, there are many metric-based or machine learning techniques ready for use [105, 187], thus automating some *MarkedAsDuplicated* and *UnMarkedAsDuplicated* tasks.

Current technologies may be less mature for helping elite developers on communicative tasks. As shown in Fig. 2.2, 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. [88] 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 automated 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 [101, 104, 177].

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 [182] 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 non-intrusive way to gain widespread acceptance. Moreover, our results also call for innovative workflow designs that does not only focusing on technical contributions (e.g., multiple-committer model in [157]) but also those optimized for the full spectrum of activities.

5.1.5 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 non-technical responsibilities.*
- *Tool support for relieving elite developers from routine administrative burdens by synthesizing existing techniques to their routine workflow.*

5.2 Empirical Studies on Automated Workflow and SE Bots

5.2.1 Increasing and Integrating Applications of SE Bot

Over 60% of the sampled repositories in Chapter 3 employed at least one bot to automate routine workflows. This high adoption rate suggests that SE bots have become prevalent in OSS projects, and have shown their increasing presence in OSS development. Compared to prior studies on SE bots in popular GitHub repository, there has been a remarkable leap in adoption rate [180]. Though bots have not been sophisticated enough to handle advanced tasks in practice, the convenience provided by bots has outweighed many of their drawbacks, e.g., disruptive notifications [156]. The current practice of OSS development on GitHub has become a semi-automated procedure that is heavily assisted by bots.

Besides, the mixed-method identification process applied in this study has the potential to improve identifying SE bots. Through validation by human efforts, we have proved that the multi-tier identification method yield improved bot identification results compared to the comment-based classifier. Future studies may consider including other tiers of input as features to determine bot activity, and therefore improve the identification techniques. Finally, this method employed in this study provided a baseline to identify bots on software engineering data set, especially GitHub platform.

However, popular bots that we identified employed similar rule-based design mechanisms. For the limited number of open-source bots, their implementations employed a rule-based system, and the system subscribed to and acted based upon certain repository events or event payloads. For example, when offering automatic issue labeling, `carsonbot` required a strictly formatted issue with content in its template. Only when the contributor filled the label entry with the pre-defined keywords, the `carsonbot` may label the issue. While many other bots employ the same mechanism and are all subscribed to GitHub's event timeline, the server's load increases exponentially as user-generated events accumulate. One engineering implication from this observation is that future bot service development may consider collecting events from one repository event-bus-like bot service infrastructure, and therefore we may reduce central server load and ensure data reliability. On the other hand, the simple rule-based design inherits the limitation in interactiveness. Similarly, other studies also argued various reasons why interactiveness has not yet been improved [1]. Its rule-based design mechanism has limited bots to respond and act under pre-defined input for completing simple tedious tasks. Thus, a future direction is to employ an advanced chat agent to empower the interaction with developers, especially in many community-related tasks such as welcoming and acknowledging contributors [152].

Another major observation is the prevalence of multi-tasking *butlers* in popular GitHub repositories, and we may anticipate a growing percentage of bots that automate multiple tasks instead of specializing in just one [51]. In addition to prior studies, this study has found an increasing number of SE bots merging various functionalities (see Figure 3.2). Software engineers have created bots by combining various existing functions. This observation suggests a trend of using SE bots in OSS repositories: assembling existing automation functions according to a project or ecosystem needs. In these repositories, SE bots have become housekeeping and multi-tasking *butlers*. For example, the two most popular ecosystem bots, `facebook-github-bot` and `Googlebot` have integrated several atomic automating functions. However, they have shown different emphases, i.e., `facebook-github-bot` focuses more on CI/CD practice while `Googlebot` focuses more on community management. As these bots somewhat reflect these organizations' ideology of developing software, important takeaways of this observation are twofold: first, software project and ecosystem have their unique requirements and preferences for software engineering process, and future SE bots may consider providing a customized space not only in functions but workflow integration; and second, creating the infrastructure of SE bots or a framework for integrating various types of existing automation has its practical value for software maintenance.

Finally, for over two-thirds (69%) of the identified bots, their implementations yet remained opaque to the public. While many bot creation frameworks are helping to create and shop simple automation, over half (53%) of existing bots were only available or applicable to specific repositories or organizations. I argue that there were two major reasons for bots to be kept private or closed-source. First, there are security and business concerns about a bot's functions. To perform various repository tasks seamlessly, SE bots often have a high level of permission (above the write access) in these repositories, and the data flows carried by these bots may be security-sensitive such as access tokens[2] and SSH keys[3]. Open-sourcing these bots may expose the vulnerabilities in a repository's automated workflow, which leads to risks of malicious behaviors. Any organization would prefer not to undertake these risks. Second, as mentioned in the previous paragraph, many customized *butler* bots integrate multiple existing services and adapt to the needs of a specific project or ecosystem, and therefore, developers have few motivations to share source code as many components of such bots have been publicly available already. However, without a framework that integrates various bot services, the current practice is far from optimal for developing SE bots as many repositories choose to withhold their bots.

5.2.2 Developer Experience and Expectation of Future SE Bots

The second interview study advances our knowledge of how expert and elite developers integrate bots into their daily SE workflow and demonstrates several challenges with the current bot-assisted workflow. Moreover, this study summarizes their experience of using bots and also their expectations of future bots. Finally, the results of this study call for

[2]Access tokens: `https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token`

[3]SSH keys: `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent`

a framework that not only manages valuable automation but also promotes OSS' sharing spirits. In addition, this study provides directions for bot usability research.

This study particularly advances our understanding of how elite developers incorporate SE bots into their software development workflow. In addition to prior studies, this study provides a full picture of how bots proactively and re-actively integrate into elite developers' full spectrum of software development activities. For technical activities, elite developers require more hands-on control over bots, i.e., explicitly configure how bots take commends and what repository events the bots should respond to. Therefore, sufficient feedback such as notifications from these bots can complement developers' awareness of the repository status [45]. However, on the other end of the spectrum, developers took another approach while handling community support, repository security, legal issues, etc: Leveraging proactive SE bots as a buffer between demands of the above aspects and developer actions. This finding provides two dimensions of implications for designing future SE bots, which advise practitioners to have different considerations while assisting elite developers.

Furthermore, this study also demonstrates the important role of notifications in software development at GitHub and similar code hosting platforms. For elite developers, notifications in CI/CD pipelines, these notifications are additional testing and deployment information, and immediate feedback provided by SE bots. On the other hand, notifications from community requests and ones produced by other proactive bots work in a different approach as reminders or to-do lists without prioritization [38]. Therefore, these two types of SE bots employing the same notification channel could have led to additional mental load and mixed presumptions of underlying tasks [172]. My empirical confirmation suggests that various categories of notifications should be delivered to developers at disparate paces. In this chapter, I argue that reactive SE bots who provided testing and deployment feedback should deliver notifications in a timely responsive, and immediate approach as they are now. Thus, they reinforce the transparent CI feedback loop and enhance the quality of software production continuously. Nevertheless, notifications from proactive SE bots can be delivered in a controllable manner which allows several degrees of customization. For instance, high-level dependency updates for security issues may yield real-time notification delivery, while weekly repository activity summaries and inactive issue reminders can be merged into a notification batch. Therefore, controlling notifications generated by bots can enhance developers' productivity while working on technical and creative tasks without additional noise [156]. Moreover, it can also improve elite developers' well-being without worrying about the project all the time but still keep them alerted about essential actions on security and privacy.

Finally, for improving the interactivity of GitHub bots, there are various comment styles. One viewpoint suggests that making bots more human-like can enhance the level of engagement and make communication more natural and relatable for humans [132]. Therefore an improved user experience and trust could be achieved from interaction [98]. However, according to Wessel et al., SE bots ought to limit their human-like interactions—phrases such as "thank you", for example—given that these interactions can be perceived as insincere and could lead to user frustration [181]. Therefore, to decide whether to make SE bots' interactions more "human-like" or not, we should balance potential drawbacks and ethical concerns.

5.3 Limitations

Similar to other empirical studies, our study is not free of threats to validity. I briefly discuss them from three perspectives.

First, from the perspective of **construct validity**, the first empirical study involves six primary constructs, which are four categories of GitHub activities, and project productivity and quality (each with two metrics, a total of four metrics). All their definitions and operationalizations are based on prior literature. For the four acting categories, I carefully follow the standard procedure to develop the mappings between raw GitHub activities and these categories. The two project outcomes are adapted from Software Engineering 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, most of the threats to construct validity have been mitigated.

Second, from the perspective of **internal validity**, there were multiple measures to ensure that the data collection process avoids the most of perils summarized in [18, 83]. For example, all subjected projects are all large ones with established governing structures 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. The main analysis method is a mature, widely used analysis technique, and empirically justifies the use of the fixed effects models in panel regressions.

One 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, of the 20 projects studied in this paper, none of them has all the channel data available. If this chapter's study applied multiple data sources for some projects but a single data source for the rest, guaranteeing 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 [20, 46, 94]. Thus, not employing multiple data sources is a major limitation, but not a serious threat to internal validity, as pointed out by Margaret-Anne Storey in her ICSE'19 keynote [155].

Third, from the perspective of **external validity**, 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 and the following studies leverage projects from five major OSS ecosystems or top popular repositories. We urge caution, however, for applying our findings in the context of small or medium size open-source projects.

Chapter 6

Concluding Remarks and Future Work

This dissertation, started and rooted in the social structure and community reputation within Open Source Software (OSS) communities, delineates a clear distinction between elite and non-elite developers based on their managerial privileges. Although the integral role of elite developers in OSS development has been recognized in software engineering literature, comprehensive investigation into their activities had been lacking until now. Drawing on fine-grained event data from 20 OSS projects, five major ecosystems, and bot usage data from the most active repositories, this dissertation provides an in-depth, dynamic portrait of elite developers' activities. These are categorized into four high-level, interpretive groups, with an analysis of their impact on project outcomes in terms of productivity and quality. Furthermore, this dissertation expands our understanding of the current workflow of elite developers with SE bots and proposes an innovative automation management framework to centralize control over automation features in development.

This dissertation offers a suite of empirical findings regarding the role of elite developers in OSS development. It affirms the pivotal contribution of elite developers, who account for the majority of activities across all four high-level categories: communicative, organizational, supportive, and typical. It further unveils a shift in elite developers' activities from *technical* work to *project management* tasks, as communicative and supportive activities escalate more rapidly than typical development activities. Notably, it establishes a significant correlation between the distribution of elite developers' efforts and project productivity and quality outcomes. These findings not only highlight the complex dynamics of elite developers' effort distribution, but also hint at a dilemma many OSS elite developers encounter: as a project grows, elite developers are compelled to undertake an increasing amount of communicative and supportive work. This dissertation further delves into the practical and design implications of these findings.

Furthermore, this dissertation sheds light on the utilization of automated techniques by OSS practitioners, particularly SE bots, within their OSS contribution workflow. Both empirical data mining and practitioner interviews attest to the importance of SE bots as integral extensions of modern OSS development teams. Nevertheless, deficiencies in interactivity and control over notification mechanisms have led to unnecessary interruptions and context-switching. To address these challenges, this dissertation introduces the Bot Management Console (BMC), an automation management framework designed to optimize the management experience of automation features and control various aspects of human-bot interactions. The effectiveness of this framework is corroborated through simulated deployment, marking a promising step towards a more efficient and seamless integration of bots in OSS development.

In continuing the research trajectory initiated by this dissertation, my future work aims to deepen the understanding of elite developers and OSS communities. As an immediate next step, I propose to extend the empirical study from the second chapter by including a larger sample of projects. This will allow us to further investigate the unique and context-dependent differences among elite developers. It's also important to recognize that project outcomes extend beyond basic productivity and quality metrics. Thus, exploring alternative outcomes, especially those related to social and human aspects like the influx of newcomers and long-term project sustainability, may offer valuable insights into the broader objectives of OSS communities. Additionally, employing diverse evaluation approaches could highlight novel design insights [74]. It would be beneficial to further evaluate the management console with practitioners, utilizing design critique sessions to obtain more nuanced feedback and identify opportunities for usability improvements [4]. Lastly, future research could explore innovative reward mechanisms and OSS contribution workflow designs. These should aim to strike a balance between technical and non-technical contributions, ensuring equitable recognition of all project members' efforts. This progressive approach promises to further enrich our understanding of OSS communities and their unique dynamics.

Bibliography

[1] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th international conference on mining software repositories*, pages 174–185, 2020.

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

[3] Emad Shihab Ahmad Abdellatif, Khaled Badran. A repository of research articles on software bots. http://papers.botse.org.

[4] Lorans Alabood, Zahra Aminolroaya, Dianna Yim, Omar Addam, and Frank Maurer. A systematic literature review of the design critique method. *Information and Software Technology*, page 107081, 2022.

[5] Maria Vaz Almeida and António Lucas Soares. Knowledge sharing in project-based organizations: Overcoming the informational limbo. *International Journal of Information Management*, 34(6):770–779, 2014.

[6] Juan Jose Amor, Gregorio Robles, and Jesus M. Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research*, EDSER '06, pages 3–6, New York, NY, USA, 2006. ACM.

[7] Le An, Marco Castelluccio, and Foutse Khomh. An empirical study of dll injection bugs in the firefox ecosystem. *Empirical Software Engineering*, 24(4):1799–1822, 2019.

[8] John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370, 2006.

[9] Maryi Arciniegas-Mendez, Alexey Zagalsky, Margaret-Anne Storey, and Allyson Fiona Hadwin. Using the model of regulation to understand software development collaboration practices and tool support. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, pages 1049–1065, New York, NY, USA, 2017. ACM.

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

[11] Sogol Balali, Igor Steinmacher, Umayal Annamalai, Anita Sarma, and Marco Aurelio Gerosa. Newcomers' barriers. . . is that all? an analysis of mentors' and newcomers' barriers in oss projects. *Computer Supported Cooperative Work (CSCW)*, 27(3):679–714, Dec 2018.

[12] Sebastian Baltes and Stephan Diehl. 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, page 187–200, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Sean Banerjee, Jordan Helmick, Zahid Syed, and Bojan Cukic. Eclipse vs. mozilla: A comparison of two large-scale open source problem report repositories. In *Proceedings of the 2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, HASE '15, pages 263–270, Washington, DC, USA, 2015. IEEE Computer Society.

[14] Gunnar R Bergersen, Dag IK Sjøberg, and Tore Dybå. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering*, 40(12):1163–1184, 2014.

[15] David M Beskow and Kathleen M Carley. Bot-hunter: a tiered approach to detecting & characterizing automated activity on twitter. In *Conference paper. SBP-BRiMS: International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation*, volume 3, page 3, 2018.

[16] Christian Bird. Sociotechnical coordination and collaboration in open source software. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 568–573. IEEE, 2011.

[17] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 24–35, New York, NY, USA, 2008. ACM.

[18] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. The promises and perils of mining git. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 1–10, 2009.

[19] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*, pages 188–197. IEEE, 2013.

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

[21] Hudson Borges and Marco Tulio Valente. What's in a GitHub star? understanding repository starring practices in a social coding platform. *JSS*, 146:112–129, 2018.

[22] Jan Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[23] François Bousquet and Christophe Le Page. Multi-agent simulations and ecosystem management: a review. *Ecological modelling*, 176(3-4):313–332, 2004.

[24] Felix C Brodbeck. Software-entwicklung: Ein tätigkeitsspektrum mit vielfältigen kommunikations-und lernanforderungen. In Felix C Brodbeck and Michael Frese, editors, *Produktivität und Qualität in Software-Projekten: Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung*, pages 13–34. Oldenbourg-Verlag, 1994.

[25] Frederick Brooks and H Kugler. *No silver bullet*. April, 1987.

[26] C Marlin Brown. *Human-computer interface design guidelines*. Intellect Books, 1999.

[27] Chris Brown and Chris Parnin. Sorry to bother you: Designing bots for effective recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 54–58. IEEE, 2019.

[28] Rune Todnem By. Organisational change management: A critical review. *Journal of change management*, 5(4):369–380, 2005.

[29] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 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, pages 44:1–44:11, New York, NY, USA, 2012. ACM.

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

[31] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 396–407, New York, NY, USA, 2017. ACM.

[32] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

[33] Benjamin Collier, Moira Burke, Niki Kittur, and Robert E Kraut. Promoting good management: Governance, promotion, and leadership in open collaboration communities. page 220, 2010.

[34] Catarina Costa, Jair Figueiredo, Anita Sarma, and Leonardo Murta. Tipmerge: recommending developers for merging branches. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 998–1002, 2016.

[35] Yves Croissant and Giovanni Millo. Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27(2):1–43, 2008.

[36] Kevin Crowston, Hala Annabi, James Howison, and Chengetai Masango. 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, pages 18–26, New York, NY, USA, 2004. ACM.

[37] Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 2005.

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

[39] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7:1–7:35, March 2008.

[40] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, HICSS '06, pages 118:1–118:10. IEEE, 2006.

[41] Daniel Alencar da Costa, Uirá Kulesza, Eduardo Aranha, and Roberta Coelho. Unveiling developers contributions behind code commits: An exploratory study. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1152–1157, New York, NY, USA, 2014. ACM.

[42] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286, 2012.

[43] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline P. de Vries. Moving into a new software project landscape. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 275–284, New York, NY, USA, 2010. ACM.

[44] Cleidson R.B. de Souza, Fernando Figueira Filho, Müller Miranda, Renato Pina Ferreira, Christoph Treude, and Leif Singer. The social side of software platform ecosystems. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3204–3214, New York, NY, USA, 2016. ACM.

[45] Tom DeMarco and Tim Lister. *Peopleware: Productive Projects And Teams*. Addison-Wesley, 2013.

[46] Martyn Denscombe. *The Good Research Guide: For Small-scale Social Research Projects*. McGraw-Hill, 2014.

[47] Luis Felipe Dias, Igor Steinmacher, and Gustavo Pinto. Who drives company-owned oss projects: internal or external members? *Journal of the Brazilian Computer Society*, 24(1):16, 2018.

[48] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. Scaling static analyses at facebook. *Commun. ACM*, 62(8):62–70, July 2019.

[49] Nicolas Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, 2005.

[50] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. *Guide to advanced empirical software engineering*, pages 285–311, 2008.

[51] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. Current and future bots in software development. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 7–11. IEEE, 2019.

[52] Linda Erlenhov, Francisco Gomes De Oliveira Neto, and Philipp Leitner. An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 445–455, 2020.

[53] Ilker Etikan, Sulaiman Abubakar Musa, and Rukayya Sunusi Alkassim. Comparison of convenience sampling and purposive sampling. *American Journal of Theoretical and Applied Statistics*, 5(1):1–4, 2016.

[54] Holly J Falk-Krzesinski, Noshir Contractor, Stephen M Fiore, Kara L Hall, Cathleen Kane, Joann Keyton, Julie Thompson Klein, Bonnie Spring, Daniel Stokols, and William Trochim. Mapping a research agenda for the science of team science. *Research Evaluation*, 20(2):145–158, 2011.

[55] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[56] Stephen M Fiore. Interdisciplinarity as teamwork: How the science of teams can inform team science. *Small Group Research*, 39(3):251–277, 2008.

[57] Kristin E Flegal and Michael C Anderson. Overthinking skilled motor performance: Or why those who teach can't do. *Psychonomic Bulletin & Review*, 15(5):927–932, 2008.

[58] Lee Fleming and David M Waguespack. Brokerage, boundary spanning, and leadership in open innovation communities. *Organization science*, 18(2):165–180, 2007.

[59] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. The space of developer productivity: There's more to it than you think. *Queue*, 19(1):20–48, 2021.

[60] Oscar Franco-Bedoya, David Ameller, Dolors Costal, and Xavier Franch. Open source software ecosystems. *Inf. Softw. Technol.*, 91(C):160–185, November 2017.

[61] R Stuart Geiger, Nelle Varoquaux, Charlotte Mazel-Cabasse, and Chris Holdgraf. The types, roles, and practices of documentation in data analytics open source software libraries. *Computer Supported Cooperative Work (CSCW)*, 27(3-6):767–802, 2018.

[62] Daniel M German, Bram Adams, and Ahmed E Hassan. The evolution of the r software ecosystem. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 243–252. IEEE, 2013.

[63] Matt Germonprez, Julie E Kendall, Kenneth E Kendall, Lars Mathiassen, Brett Young, and Brian Warner. A theory of responsive design: A field study of corporate engagement with open source communities. *Information Systems Research*, 28(1):64–83, 2016.

[64] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. In *Proc. ICSE*, pages 1046–1058, 2021.

[65] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. A ground-truth dataset and classification model for detecting bots in github issue and pr comments. *Journal of Systems and Software*, 175:110911, 2021.

[66] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. 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, page 358–368. IEEE Press, 2015.

[67] Michaela Greiler, Margaret-Anne Storey, and Abi Noda. An actionable framework for understanding and improving developer experience. *IEEE Transactions on Software Engineering*, 2022.

[68] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K Heinz, Geir Huse, et al. A standard protocol for describing individual-based and agent-based models. *Ecological modelling*, 198(1-2):115–126, 2006.

[69] Volker Grimm, Uta Berger, Donald L DeAngelis, J Gary Polhill, Jarl Giske, and Steven F Railsback. The odd protocol: a review and first update. *Ecological modelling*, 221(23):2760–2768, 2010.

[70] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. "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, page 395–404, New York, NY, USA, 2011. Association for Computing Machinery.

[71] Marvin Hanisch, Carolin Haeussler, Stefan Berreiter, and Sven Apel. Developers' progression from periphery to core in the linux kernel development project. In *Academy of Management Proceedings*, volume 2018, page 14263. Academy of Management Briarcliff Manor, NY 10510, 2018.

[72] Geir K Hanssen. A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. *Journal of Systems and Software*, 85(7):1455–1466, 2012.

[73] James D Herbsleb. Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering (FOSE'07)*, pages 188–198. IEEE, 2007.

[74] David M Hilbert and David F Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys (CSUR)*, 32(4):384–421, 2000.

[75] Youyang Hou and Dakuo Wang. Hacking with npos: Collaborative analytics and broker roles in civic data hackathons. *Proc. ACM Hum.-Comput. Interact.*, 1(CSCW):53:1–53:16, December 2017.

[76] James Howison and Kevin Crowston. Collaboration through open superposition: a theory of the open source way. *Management Information Systems Quarterly*, 38(1):29–50, 2014.

[77] Federico Iannacci. Coordination processes in open source software development: The linux case study. *Emergence: Complexity & Organization*, 7(2):21–31, 2005.

[78] Slinger Jansen, Michael A Cusumano, and Sjaak Brinkkemper. *Software ecosystems: analyzing and managing business networks in the software industry.* Edward Elgar Publishing, 2013.

[79] Chris Jensen and Walt Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 364–374, Washington, DC, USA, 2007. IEEE Computer Society.

[80] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 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, page 70–80, New York, NY, USA, 2011. Association for Computing Machinery.

[81] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 164–174, Piscataway, NJ, USA, 2017. IEEE Press.

[82] Nicolas Jullien, Klaas-Jan Stol, and James Herbsleb. A preliminary theory for open source ecosystem micro-economics. *arXiv preprint arXiv:1905.05985*, 2019.

[83] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR '14, page 92–101, New York, NY, USA, 2014. Association for Computing Machinery.

[84] Hanna Kallio, Anna-Maija Pietilä, Martin Johnson, and Mari Kangasniemi. Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *Journal of advanced nursing*, 72(12):2954–2965, 2016.

[85] Mücahit Kantepe and Murat Can Ganiz. Preprocessing framework for twitter bot detection. In *2017 International conference on computer science and engineering (ubmk)*, pages 630–634. IEEE, 2017.

[86] Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. 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, page 179–188. IEEE Press, 2012.

[87] Terhi Kilamo, Imed Hammouda, Tommi Mikkonen, and Timo Aaltonen. From proprietary to open source—growing an open source ecosystem. *Journal of Systems and Software*, 85(7):1467–1478, 2012.

[88] Dongsun Kim, Yida Tao, Sunghun Kim, and Andreas Zeller. Where should we fix this bug? a two-phase recommendation model. *IEEE Transactions on Software Engineering*, 39(11):1597–1610, 2013.

[89] Sunghun Kim and E. James Whitehead. How long did it take to fix bugs? In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, page 173–174, New York, NY, USA, 2006. Association for Computing Machinery.

[90] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering*, 28(8):721–734, 2002.

[91] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, VLHCC '04, pages 199–206, Washington, DC, USA, 2004. IEEE Computer Society.

[92] Stefan Koch and Georg Schneider. Effort, co-operation and co-ordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[93] Ellen Ernst Kossek, Karen Roberts, Sandra Fisher, and Beverly Demarr. Career self-management: A quasi-experimental assessment of the effects of a training intervention. *Personnel Psychology*, 51(4):935–960, 1998.

[94] Chakravanti Rajagopalachari Kothari. *Research Methodology: Methods and Techniques*. New Age International, 2004.

[95] Sara Landset, Taghi M Khoshgoftaar, Aaron N Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):24, 2015.

[96] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[97] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. Software bots. *IEEESoft*, 35(1):18–23, 2018.

[98] John D Lee and Katrina A See. Trust in automation: Designing for appropriate reliance. *Human factors*, 46(1):50–80, 2004.

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

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

[101] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 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)*, pages 110–119, 2016.

[102] Bin Lin, Gregorio Robles, and Alexander Serebrenik. 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, pages 66–75. IEEE, 2017.

[103] Dongyu Liu, Micah J Smith, and Kalyan Veeramachaneni. Understanding user-bot interactions for small-scale automation in open-source development. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2020.

[104] Fei Liu, Jeffery Flanigan, Sam Thomson, Norman Smith, and Noah A. Sadeh. 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)*, pages 1077–1086, 2015.

[105] Dmitry V Luciv, Dmitrij V Koznov, George A Chernishev, Andrey N Terekhov, K Yu Romanovsky, and Dmitry A Grigoriev. Detecting near duplicates in software documentation. *Programming and Computer Software*, 44:335–343, 2018.

[106] Mircea Lungu, Michele Lanza, Tudor Gîrba, and Romain Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, 75(4):264–275, 2010.

[107] Wanwangying Ma, Lin Chen, Xiangyu Zhang, Yuming Zhou, and Baowen Xu. How do developers fix cross-project correlated bugs?: A case study on the github scientific python ecosystem. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 381–392, Piscataway, NJ, USA, 2017. IEEE Press.

[108] Kathleen M MacQueen, Eleanor McLellan, Kelly Kay, and Bobby Milstein. Codebook development for team-based qualitative analysis. *Cam Journal*, 10(2):31–36, 1998.

[109] Konstantinos Manikas and Klaus Marius Hansen. Software ecosystems–a systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306, 2013.

[110] Linda Mannila, Mia Peltomäki, and Tapio Salakoski. What about a simple language? analyzing the difficulties in learning to program. *Computer Science Education*, 16(3):211–227, 2006.

[111] Jennifer Marlow and Laura Dabbish. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, page 145–156, New York, NY, USA, 2013. Association for Computing Machinery.

[112] David W McDonald and Mark S Ackerman. Just talk to me: a field study of expertise location. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 315–324, 1998.

[113] David W McDonald and Mark S Ackerman. Expertise recommender: a flexible recommendation system and architecture. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 231–240. ACM, 2000.

[114] Katherine B McKeithen, Judith S Reitman, Henry H Rueter, and Stephen C Hirtle. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13(3):307–325, 1981.

[115] Nenad Medvidovic and Richard N Taylor. Software architecture: foundations, theory, and practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 471–472, 2010.

[116] André N Meyer, Laura E Barton, Gail C Murphy, Thomas Zimmermann, and Thomas Fritz. The work life of developers: Activities, switches and perceived productivity. *TSE*, 43(12):1178–1193, 2017.

[117] Nelson Minar, Roger Burkhart, Chris Langton, Manor Askenazi, et al. The swarm simulation system: A toolkit for building multi-agent simulations. 1996.

[118] Shawn Minto and Gail C Murphy. Recommending emergent teams. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, pages 5–5. IEEE, 2007.

[119] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[120] Audris Mockus and James D. Herbsleb. Expertise browser: A quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, page 503–512, New York, NY, USA, 2002. Association for Computing Machinery.

[121] Nigel Nicholson. A theory of work role transitions. *Administrative science quarterly*, 29(2):172–191, 1984.

[122] Gina Colarelli O'Connor, Mark P Rice, Lois Peters, and Robert W Veryzer. Managing interdisciplinary, longitudinal research teams: Extending grounded theory-building methodologies. *Organization Science*, 14(4):353–373, 2003.

[123] S. Okon and S. Hanenberg. Can we enforce a benefit for dynamically typed languages in comparison to statically typed ones? a controlled experiment. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10, May 2016.

[124] Gary M Olson and Judith S Olson. Distance matters. *Human–computer interaction*, 15(2-3):139–178, 2000.

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

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

[127] Fernando Perez, Brian E Granger, and John D Hunter. Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21, 2010.

[128] Marian Petre and André van der Hoek. Beyond coding: Toward software development expertise. *XRDS: Crossroads, The ACM Magazine for Students*, 25(1):22–26, 2018.

[129] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[130] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 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 2014, page 155–165, New York, NY, USA, 2014. Association for Computing Machinery.

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

[132] Byron Reeves and Clifford Nass. The media equation: How people treat computers, television, and new media like real people. *Cambridge, UK*, 10:236605, 1996.

[133] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. 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, pages 541–550, New York, NY, USA, 2008. ACM.

[134] Romain Robbes, Mircea Lungu, and David Röthlisberger. How do developers react to api deprecation?: The case of a smalltalk ecosystem. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 56:1–56:11, New York, NY, USA, 2012. ACM.

[135] Peter Robe, Sandeep K Kuttal, Jake AuBuchon, and Jacob Hart. Pair programming conversations with agents vs. developers: challenges and opportunities for se community. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 319–331, 2022.

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

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

[138] Paul A Ruud. Tests of specification in econometrics. *Econometric Reviews*, 3(2):211–242, 1984.

[139] Anita Sarma, Xiaofan Chen, Sandeep Kuttal, Laura Dabbish, and Zhendong Wang. Hiring in the global stage: Profiles of online contributions. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, pages 1–10, Aug 2016.

[140] Walt Scacchi. Socio-technical design. *The Encyclopedia of Human-Computer Interaction*, 1:656–659, 2004.

[141] Walt Scacchi. 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, pages 459–468, New York, NY, USA, 2007. ACM.

[142] Walt Scacchi, Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani. Understanding free/open source software development processes, 2006.

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

[144] Klaus-Benedikt Schultis, Christoph Elsner, and Daniel Lohmann. Architecture challenges for internal software ecosystems: a large-scale industry case study. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 542–552, 2014.

[145] Pankaj Setia, Balaji Rajagopalan, Vallabh Sambamurthy, and Roger Calantone. How peripheral developers contribute to open-source software development. *Info. Sys. Research*, 23(1):144–163, March 2012.

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

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

[148] Robert Michael Siegfried, Jason Siegfried, and Gina Alexandro. A longitudinal analysis of the reid list of first programming languages. *Information Systems Education Journal*, 14(6):47–54, 2016.

[149] John Skvoretz. Complexity theory and models for social networks. *Complexity*, 8(1):47–55, 2002.

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

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

[152] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 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, page 1379–1392, New York, NY, USA, 2015. Association for Computing Machinery.

[153] Daniel Stokols. People in places: A transactional view of settings. *Cognition, social behavior, and the environment*, pages 441–488, 1981.

[154] Daniel Stokols. *Social ecology in the digital age: Solving complex problems in a globalized world.* Academic Press, 2018.

[155] Margaret-Anne Storey. 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, pages 2–2, Piscataway, NJ, USA, 2019. IEEE Press.

[156] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proc. FSE*, page 928–931, 2016.

[157] Xin Tan. 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, page 1205–1207, New York, NY, USA, 2019. Association for Computing Machinery.

[158] Richard N Taylor, Nenad Medvidovic, and Eric Dashofy. *Software Architecture: Foundations, Theory, and Practice.* John Wiley & Sons, 2009.

[159] Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Anita Sarma, and Igor Steinmacher. Hidden figures: Roles and pathways of successful oss contributors. *Proceedings of the ACM on human-computer interaction*, 4(CSCW2):1–22, 2020.

[160] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. Adding sparkle to social coding: An empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 511–522, New York, NY, USA, 2018. ACM.

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

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

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

[164] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark G.J. van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 3789–3798, New York, NY, USA, 2015. Association for Computing Machinery.

[165] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. On the variation and specialisation of workload—a case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, 2014.

[166] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 805–816, 2015.

[167] Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo, Pooyan Jamshidi, and Christian Kästner. Understanding similarities and differences in software development practices across domains. In *Proceedings of the 14th International Conference on Global Software Engineering*, ICGSE '19, pages 74–84, Piscataway, NJ, USA, 2019. IEEE Press.

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

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

[170] Patrick Wagstrom, Corey Jergensen, and Anita Sarma. 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, 2012.

[171] Zhendong Wang, Yang Feng, Yi Wang, James A Jones, and David Redmiles. Elite developers' activities at open source ecosystem level. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, pages 312–313, 2020.

[172] Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. Unveiling elite developers' activities in open source projects. *ACM Trans. Softw. Eng. Methodol.*, 29(3), June 2020.

[173] Zhendong Wang, Yi Wang, and David Redmiles. Competence-confidence gap: a threat to female developers' contribution on github. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 81–90. IEEE, 2018.

[174] Zhendong Wang, Yi Wang, and David Redmiles. From specialized mechanics to project butlers: the usage of bots in OSS development. *IEEE Software*, 2022.

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

[176] E. F. Weller. Practical applications of statistical process control [in software development projects]. *IEEE Software*, 17(3):48–55, May 2000.

[177] Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 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)*, pages 1711–1721, 2015.

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

[179] Mairieli Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco A Gerosa, and Igor Steinmacher. Bots for pull requests: The good, the bad, and the promising. In *Proceedings of the 44th International Conference on Software Engineering*, pages 274–286, 2022.

[180] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–19, 2018.

[181] Mairieli Wessel, Andy Zaidman, Marco A Gerosa, and Igor Steinmacher. Guidelines for developing bots for github. *IEEE Software*, 2022.

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

[183] Jeffrey M. Wooldridge. *Introductory Econometrics: A Modern Approach*. Cengage Learning, 5 edition, 2012.

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

[185] Svetlana Yarosh, Tara Matthews, Michelle Zhou, and Kate Ehrlich. I need someone to help! a taxonomy of helper-finding activities in the enterprise. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1375–1386, 2013.

[186] Dawit Yimam-Seid and Alfred Kobsa. Expert-finding systems for organizations: Problem and domain analysis and the demoir approach. *Journal of Organizational Computing and Electronic Commerce*, 13(1):1–24, 2003.

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

[188] Yuxia Zhang, Minghui Zhou, Klaas-Jan Stol, Jianyu Wu, and Zhi Jin. How do companies collaborate in open source ecosystems? an empirical study of openstack. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1196–1208. IEEE, 2020.

[189] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. On the scalability of linux kernel maintainers' work. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE '17, page 27–37, New York, NY, USA, 2017. Association for Computing Machinery.

Appendix A

SE Bot User Study

A.1 Solicitation Message for Private Organizations

Hello, my name is Zhendong Wang and I am a doctoral candidate in the Informatics Department at the University of California, Irvine. I am conducting a research study to understand and improve open source developers' experience with automation technologies (such as experience with GitHub bots). This work has the potential to improve the productivity and sustainability of open source projects. I am recruiting developers who have the administrative privilege (write-access in the main repository) in large-scale open source projects to participate in an interview study.

Participation in this study will take approximately 20 to 30 minutes via Zoom. Participation in this study is fully voluntary. You are free to choose not to participate or withdraw from the study at any time.

We deeply respect the privacy of all participants, and guarantee complete anonymity throughout your participation. The data will be kept secure on a private hard disk and password protected. No personal identifiers will be kept. You will receive $20 Amazon gift card for participating in this study. In addition, based on your response, you may be invited to another study. If you are willing to participate in this study, please fill out the following form:

[*expired intent form link*]

We plan to include the results of the study in a scientific publication. Should you be interested in being informed about the outcome of this study, you may indicate so by providing us with an e-mail address in the form. Please feel free to contact me for any questions about the study at:

zhendow@uci.edu

A.2 SE Bot Usage Participant Intent Form

Participant intent response form for a study of SE bots on GitHub. If there were any questions about this study please do not hesitate to contact Zhendong Wang at: zhendow@uci.edu

Are you willing to participate this interview study about automation usage at GitHub?

Yes ____________

No ____________

What is your email address so that we can contact you for scheduling the interview (your personal identifies will be deleted after the interview)?

____

Will you be comfortable to participant in the interview via Zoom in a recorded session?

Yes ____________

No ____________

If you are willing to participant in this study, please check all the following apply to you:

_____ an active open source contributor

_____ obtained write access in some GitHub repository

_____ contributed to some repositories that employed automation, such as stale bot, GitHub actions, or other automated assistance

_____ over 18

_____ able to fluently communicate in English

If the results of this study are published, would you like a copy of the initial publication?

Yes ____________

No ____________

If yes, could you provide the email address for us to send the copy:

____

A.3 SE Bot Usage Interview Protocol

Interview Protocol Project: Assisting the Elite-driven Open Source Development through Activity Data and Automation

Time of interview ____________ Date ____________ Place ____________

Interviewer ____________ Interviewee ID ____________

A.3.1 Introduction

My name is Zhendong Wang, a Ph.D. candidate in the Software Engineering program at the University of California, Irvine. Thanks for participating in this study. My research interest focuses on leveraging development activity data to support the expert and elite group of developers in the open-source community. To understand how to assist open source developers with their daily routine development activities in the open source project, I would like to ask you some questions in this interview study. Before we enter the study, I would like to provide some information about this study and some terminology that will be used by us.

We invite you to this study as you are an elite developer in the [XXX] project, who holds the administrative privilege, and you are still actively involved in contributing to this project.

The intention of this study is to understand elite developers' daily workflow, the types of activities they do, and how to support their work while mitigating the interruptions in their daily workflow. In our study, small-scale automation and bots, are specifically referred to as software agents that perform small maintenance-related tasks on repositories, serving as a conduit between users and services. For example, first-timers, issue-label-bot, and mention-bot so on.

A.3.2 Informed Consent

The interviewer will read essential content in the informed consent reviewed by IRB and remind the participant that the soundtrack of the interview would be recorded, and their responses will be kept confidential without any identity information. Besides, they may choose to quit the study at any time.

A.3.3 Developer and Project Background

This section investigates the interviewee's background information on their roles and responsibilities in the project, and also looks for project-specific detailed information. First, I would like to thank you for your contribution to the open-source community. We would like to know a bit more about you and your projects. We identified you as our target participants since we found that you have performed some administrative activity in your repository, thus,

Q1: Can you briefly introduce yourself, and the project that you contributed to?

Q2: Can you describe the routine activities you do during a typical session that you contribute to the project?

A.3.4 Non-coding Activities

This section investigates the participant's focus shifts and additional non-technical burdens as the project become mature, especially when external developers enter the project.

Thanks for answering the questions about your contributions and your background

Q3: As your project develops over time, what types of contributing activities do you enjoy?

Q4: Have you or your project employed any measures to assist or support your managerial activities?

Q5: What is your experience with these measures and technologies?

A.3.5 Bot Experience and Expectation

This section seeks answers for the first questions and focuses on the inquiries on the bots that deployed in interviewee's code repository. There are some major categories of bots in the OSS development, for example, acknowledging contributors, welcoming new contributors, finding best pull-requests reviewers, and tracking todo items etc. We would like to know,

Q6: Currently, does your repository employ any types of bot helping with

your daily activities?

Q7a: (If Q6 yes) What types of development activities in your repositories have assistance from bots? What is your experience with these bots?

Q7b: (If Q6 no) What was the reason that kept you from using bots? (Ask followup questions to let participants specify)

Q8: What are your expectations of the future automation assistance in open source development?

A.3.6 Final

This section invites participants to future design critics' follow-ups.

Q9: We are designing a set of automation to assist open source developers with their non-coding activities, will you be willing to participate in a follow-up to critique the design in the future?

Q10: Would you like to receive an early report of this study in the form of a technical report?

This is the end of the interview. Thank the individual for participating in this interview. Assure them the confidentiality of the response and potential future interviews.

Appendix B

Empirical Data Repository

B.1 Elite Developer Empirical Study

The Google Drive repository includes

- Raw event data from 2016 to 2018, collected from 20 OSS projects in Chapter 2
- Intermediate analysis data for each research question from $\mathbf{RQ_{2.1}}$ to $\mathbf{RQ_{2.3}}$

`https://drive.google.com/drive/folders/10ibmz2svPRf3jfRtm7mbiouo9ATaYAoB`

B.2 GitHub Bot Usage Study

This GitHub repository includes

- All sampled 1,000 GitHub SDE projects in Chapter 3
- A manual bot identification/review guide

- All identified bot services

`https://github.com/zhendow/GitHub_Bot_Usage`