Bringing Open Source Communication and Development Together: A Cross-Platform Study on Gitter and GitHub

Hanzhi Jiang*, Lin Shi*, Meiru Che, Yuxia Zhang, Qing Wang

Abstract—Recently, a growing body of research has realized that live chat via modern communication platforms plays an increasingly important role in OSS (Open Source Software) collaborative development. Among these platforms, Gitter has emerged as a popular choice since it is directed toward GitHub projects by account sharing and activity subscribing. But little is known about how Gitter affects the OSS development on GitHub. Who are the developers being active in both social and technical platforms? How important are they? In this paper, we perform a comprehensive cross-platform study on Gitter and GitHub, two representative platforms for live communication and distributed development, to explore the characteristics of cross-platform contributors (CPCs) and whether live chat can provoke open source development. This study yields interesting findings: 1) Despite CPCs being small in quantity yet account for a much bigger amount of communication and development; 2) Gitter continually attracts new contributors; 3) Communication on Gitter has a positive impact on the contributions of OSS developers; and 4) Inactive developers on GitHub still participate in discussions on Gitter. Based on our findings, we provide recommendations for OSS communities and developers and shed light on future research directions. We believe that the findings and insights will inspire the OSS communities, enable a broader view of the interplay between Gitter and GitHub, and enhance the sustainability of the OSS ecosystem.

Index Terms—Live chat, Team communication, Open source, Empirical study

1 Introduction

Open source software (OSS) contributors maintain a diversity of contributing activities on multiple platforms. For example, they not only commit source code via version control systems, such as GitHub, but also exert significant efforts in communication via various platforms, discussing bugs and solutions, new features, progress management, and so on [1], [2], [3], [4]. More than ever, live communication platforms, such as Gitter, Discord, Slack, and X (formerly Twitter) play a fundamental role in OSS communications and collaboration. As one type of synchronous textual communication among a community of developers, live chat allows developers to receive real-time responses from others, replacing traditional asynchronous communication like emails in some cases [1], [5], [6], [7], [8], [9], [10]. Among the aforementioned live communication platforms, Gitter is widely used by OSS contributors in terms of its openness, provided services and how it organizes chatrooms. Gitter's chatrooms are project-oriented and are open for everyone to participate easily, regardless of the level of professionality, *e.g.*, beginners are capable of seeking help from project developers to improve their project knowledge and programming skills. OSS practitioners can sign in Gitter simply using their GitHub accounts directly and refer to GitHub issues/pull requests in the utterance easily through the auto-link function.

In the literature of OSS, most studies analyze the OSS community by observing the activities of contributors from a single-platform perspective. For example, existing studies take GitHub, the most representative hosting platform for open-source projects, as the individual research subject to learn the activities [11], [12], [13], [14] and contributions [15], [16], [17], [18] of the OSS developers. While, the OSS community is not only reflected by GitHub, besides which developers' communication is also a vital part of the OSS community. Discussions about technical problems, domain knowledge, and project coordination are also important contributions to the OSS community. Only considering the single-platform analysis might overlook the communication aspect of the OSS community, leading to an insufficient understanding and measurement of developers' contributions. Moreover, there are OSS developers (cross-platform contributors, CPCs) who share knowledge and solutions to problems on communication platforms, as well as write code and report bugs on GitHub. It remains unclear what role they play in the OSS community. Understanding these questions would provide a broader view of how platforms can serve the sustainable evolution of OSS ecosystems.

- *H. Jiang is with the State Key Laboratory of Intelligent Game, Institute of Software Chinese Academy of Sciences, Beijing, China, and also with the University of Chinese Academy of Sciences, Beijing, China. E-mail: hanzhi2021@iscas.ac.cn*
- *L. Shi is with Beihang University, Beijing, China. E-mail: shilin@buaa.edu.cn*
- *M. Che is with the Data61, CSIRO Australia, Eveleigh, Australia. E-mail: Meiru.Che@data61.csiro.au*
- *Y. Zhang is with the Beijing Institute of Technology, Beijing, China. E-mail: yuxiazh@bit.edu.cn*
- *Q. Wang is with the State Key Laboratory of Intelligent Game, Institute of Software Chinese Academy of Sciences, Beijing, China, and with the University of Chinese Academy of Sciences, Beijing, China, and also with the State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China. E-mail: wq@iscas.ac.cn*

**Both authors contributed equally to this research.*
Corresponding author: Qing Wang

To bridge the gap, in this paper, we perform an in-depth cross-platform analysis on Gitter and GitHub. Seven popular open source communities that both use Gitter and GitHub platforms are selected as our studied subjects, which leads to 1,546,127 utterances from 37,060 chatting developers on Gitter, and 395,664 development activities contributed by 89,858 contributors on GitHub. By cross-linking accounts between the two platforms of a specific project, a list of 4,506 contributors who collaborate on GitHub and chat on Gitter, referred as cross-platform contributors (CPCs), is collected. This cross-platform dataset is publicly available to aid further research.[1]

To address our research concerns, we investigate traits of CPCs in terms of roles and contribution, communication preference, and behavioral consistency, as well as the promotive effect brought by live chat on OSS contribution by new contributors, onboarded contributors and returned code contributors. A variety of analysis techniques are applied: (1) onion model [19] to characterize roles and communication preferences of CPCs; (2) social network analysis to measure the social influence of CPCs; (3) correlation analysis to assess the behavioral consistency; (4) open card sort to explore the impact of live chat on attracting new contributors; (5) inactivity identification [20] and statistic analysis to validate the impact of live chat on return/contribution.

Among these results, we find that:

- Although the number of CPCs is small, which takes 12.2% on Gitter and only 5% on GitHub, their contribution is significant. The core developers in CPCs only account for 0.04% GitHub population but contribute to 19% commits.
- CPCs' communication: core developers are more likely to discuss advanced topics with a high difficulty level, while peripheral OSS developers have more social chatting on unwanted behaviors and errors.
- Though Some contributors have a low social influence level on GitHub, they can be highly influential on Gitter.
- Gitter continually attracts new contributors. Nearly 60% are OSS product users and 38% are developers in new contributors, and core developers are one of the motivating factors.
- OSS contributors who have communicated on Gitter significantly contribute more to the open source software, than those who do not communicate on Gitter.
- Inactive code developers on GitHub still participate in discussions on Gitter, providing a higher probability of returning to active code developers.

The remainder of this article is structured as follows. Section 2 introduces the two studied platforms and the reason for choosing them. Section 3 presents the description of our overall study design, research questions, and dataset. Sections 4-7 introduce our empirical approaches and the results of the four research questions, respectively. Section 8 discusses the implications and suggestions for individual developers, OSS communities, and OSS researchers. This is followed by the threats to validity in Section 9 and related work in Section 10. Then Section 11 concludes the article and sheds light on future work.

1. https://github.com/CrossPlatform2023/CrossPlatform2023

2 The Studied Platforms

2.1 OSS Development Platforms

OSS developers often rely on distributed version control systems such as GitHub, GitLab, and Sourceforge, to collaboratively contribute to OSS communities. Among them, GitHub is a dominant open-source hosting platform based on Git. The widespreadness of Git, and the flexible collaborative processes, facilitate contributing to, starting up, and migrating to team projects in GitHub [12]. GitHub follows a fork-and-pull model [17] where developers can fork the repository to make modifications in a new branch and create a pull request to ask for a merge into the original project if they want to contribute. Then further discussion and code review take place on the GitHub platform in order to assure the correctness and feasibility of code changes in their commits, after which the pull request will be either rejected or closed. Participants are also supported to report bugs and new features by submitting issue reports on GitHub.

Although GitHub serves professionally as a project management and development coordination system, it is limited in respect of making social connections for developers [21]. Despite commenting on issues and pull requests, some developers deem GitHub lacks ways to interact with others when it comes to detailed discussions such as seeking help for encountered developmental problems and knowledge sharing. This motivates us to draw a bigger picture of OSS project communities by investigating not only collaboration activities on GitHub but also the communication of developers off the GitHub platform, as well as the interplay between them.

2.2 OSS Communication Platforms

To fulfill the communication needs of massive OSS practitioners, communication platforms such as mailing lists, Gitter, Slack, Discord, X (formerly Twitter), and Microsoft Teams have emerged to facilitate knowledge-sharing and help-seeking. These communication platforms can be divided into two categories: asynchronous communication platforms and synchronous communication platforms. Asynchronous communication platforms include email-based mailing lists, question-answer-based services (*e.g.* Stack Overflow), social media-based channels (*e.g.* X), issue-based platforms (GitHub discussions), and so on. These platforms are successful in performing their determined functions, but they lack real-time response and close interaction with other developers.

Prior studies have manifested the trend of shifting from traditional asynchronous communication such as mailing lists towards instant communication chatrooms such as Gitter, Slack, Discord, and IRC [1], [4], [5], [6], [7], [8], [9], [10], [22], [23], [24]. These platforms facilitate instant communication of developers, helping developers who are distributed across multiple locations to coordinate their contribution within a community, hence enabling better team collaboration, group awareness, project coordination, and new technology sharing [8], [25]. Also, users can raise problems they encounter such as API usage and unwanted behaviors [2], acquiring closer access to development team members or experts to answer their questions. In a large-scale survey conducted by Mezouar *et al.* [26], the quality of the help received within a short response time is a major purpose of using Slack and

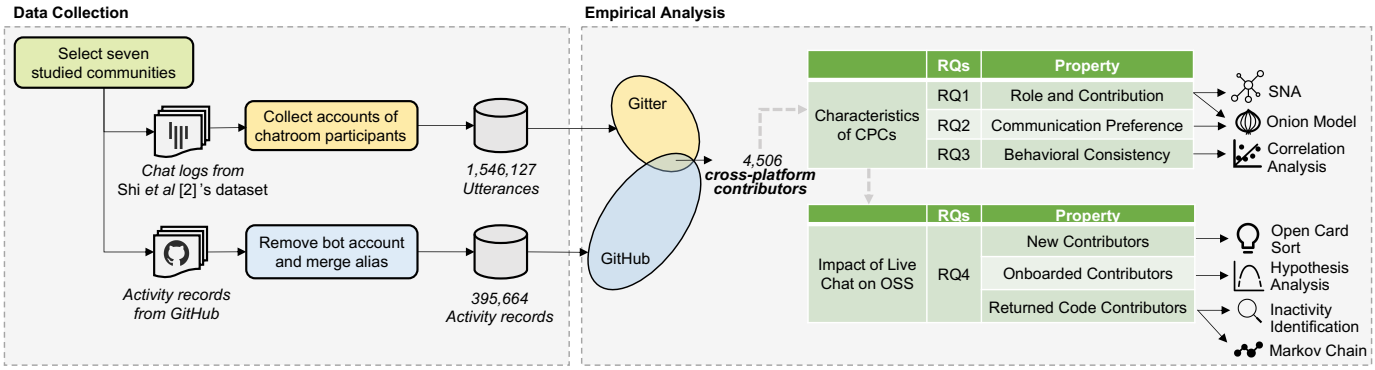


Fig. 1: Overview of our research methodology

Gitter chatrooms. Moreover, they reported the perceived impacts of using these platforms include supporting the issue resolution process, providing access to information, and brainstorming features. Therefore, these communication platforms play a vital role in the sound development of OSS, bridging the gap between multiple software tools such as code hosting platforms and communication channels, and shaping software development activities and practices [24].

However, despite the benefits investigated by prior research, we aim to revisit OSS communication along with development platforms for the following two reasons: (1) the emergence of OSS communication platforms brings cross-platform contributors, but little is known about this special group of OSS practitioners; (2) the quantitative impact of contributors' communication on software development still needs more in-depth exploration.

Here we introduce the three most representative instant communication platforms: **Gitter**, **Slack**, and **Discord**, respectively, and explain the reason for choosing Gitter as our subject communication platform. **Gitter** is a representative and prevalent instant communication platform with over 800K users, 90K communities, and 300K chatrooms. Gitter provides live chat services including code tagging, user tagging, and integrations with code hosting platforms such as GitHub, therefore, many GitHub projects are eager to display their Gitter badges as an indication of communication support [27]. **Slack** also supports developers' live chat with an intent to reduce reliance on email for internal discussions. Compared to Slack, Gitter provides more openness to those who are eager to participate in OSS communication, enabling a lower threshold to find, join, and use thus a larger population of participants [26]. **Discord** is also a popular choice when it comes to the adoption as an instant messaging tool for GitHub projects [28], [29], [30]. It enables communication through various approaches, including voice calls, video calls, text messaging, and media and files. Compared with Discord's general use in multiple situations, *e.g.*, games, cooking, and study group, Gitter is designed for open-source developers to chat. With the equipped GitHub-supporting services like account sharing and automatic issue linking in utterances, developers can easily sign in to Gitter chatrooms with their GitHub accounts and quote GitHub issues in their utterances. This helps to build a strong community of developers compared to other discussion forums.

Since our research concerns are highly relevant to the impact of live chat on the OSS project, and that Gitters' openness and tight integration with GitHub enable a wider range of contributors to investigate and save our effort in mapping accounts, we choose Gitter as the representative communication platform to conduct the cross-platform study.

3 Study Design

Overall, the research methodology consists of two phases, as illustrated in Figure 1. First, in the data collection phase, a large scale of chat utterances from seven OSS communities are collected from the Gitter dataset built in our previous work [2]. By collecting the associated development data via the shared GitHub accounts, we extend this Gitter dataset into a cross-platform dataset. Second, in the empirical analysis phase, we first investigate the characteristics of CPCs with respect to their role and contribution, communication preference, and behavioral consistency, and then we investigate the impact of instant communication on OSS contribution in terms of new contributors, onboarded contributors as well as returned code contributors.

3.1 Research Questions

Focusing on the relationship between Gitter and GitHub, we mainly investigate the characteristics of CPCs and the impact of live chat on OSS development. Specifically, this study aims to investigate the following four research questions:

- **RQ1 (CPCs' Role and Contribution)**: *Who are the developers being active in both Gitter and GitHub, and how important are they?* This research question aims to unveil the roles of developers being active in both Gitter and GitHub and examine their importance to the OSS community.
- **RQ2 (CPCs' Communication Preference)**: *What topics do CPCs in different roles prefer on Gitter?* This research question dives into CPCs' discussion contents to explore the relationship between OSS development roles and communication topics.
- **RQ3 (CPCs' Behavioral Consistency)**: *Do CPCs behave consistently across Gitter and GitHub?* This research question is designed to examine the behavioral consistency of CPCs across different platforms, with respect to their activeness, influence, and collaboration.
- **RQ4 (Gitter's Impact on OSS Contribution)**: *How does Gitter affect OSS contribution on GitHub?* This research question targets to identify how Gitter affects the OSS contribution on GitHub, in terms of new contributors,

TABLE 1: The statistics of our dataset

Community	Domain	GitHub Repository	Gitter population		GitHub population		CPC
			Participants	Utterances	Participants	Activity Records	
Angular	Frontend Framework	angular/angular	10,389	763,173	23,682	70,660	1,930
Appium	Mobile	appium/appium	1,955	31,656	6,085	25,151	280
DL4J	Data Science	deeplearning4j/deeplearning4j	3,609	266,577	2,127	29,215	756
Docker	DevOps	moby/moby	2,138	30,159	13,979	83,104	104
Ethereum	Blockchain	ethereum/go-ethereum	10,349	94,852	6,957	27,753	240
Nodejs	Web Application Framework	nodejs/node	3,803	90,336	21,708	78,616	962
Typescript	Programming Language	microsoft/TypeScript	4,817	269,374	15,320	81,165	234
	Total		37,060	1,546,127	89,858	395,664	4,506

onboarded contributors,[2] and returned code contributors.

3.2 Dataset

We build our cross-platform dataset based on our previous work [2]. In this work, we collected chat utterances of eight most-participated Gitter communities from eight active domains, covering front-end framework, mobile, data science, DevOps, blockchain platform, collaboration, web app, and programming language. Each chat log contains a sequential set of utterances (which is referred to as *dialog*) in chronological order. The poster and the posting time of each utterance are also recorded. Dialogs are disentangled in both manual and automatic ways, including 749 manually disentangled dialogs and 173,278 automatically disentangled dialogs using the FF approach [31] after assessing 4 state-of-the-art dialog disentanglement tools on the manually labeled data.

We select seven out of eight OSS communities from this prior dataset by excluding the one that does not host source code on GitHub. The selected communities include Angular,[3] Appium,[4] DL4J,[5] Docker,[6] Ethereum,[7] Nodejs,[8] and Typescript.[9] The corresponding domains and GitHub repositories are shown in Table 1.

For Gitter data, we extend the Gitter utterances provided in the previous dataset to "2022-09-25" using Gitter REST API [32] as well as the data processing mechanisms provided by them. Then, we collect accounts of utterance posters in the chatroom.

For GitHub data, we leverage GitHub REST API [33] to obtain activity records including commits, issue reports, and pull requests. Each one of them contains the following information: activity conductor, time, activity description, and the id-number. Commenters' information (commenting time, username) is recorded in issues and pull requests as well. Additionally, pull requests also contain their reviewers and related commits. The GitHub data starts from the repository initialization and is as of "2022-09-25" as well. When processing the data from GitHub, we first remove automatic bot accounts using GitHub API[10] and the classification model proposed by Golzadeh *et al.* [34] that achieves an F1-score of 0.98. The reason for bot removal is that bots are reported to perform massive maintenance tasks on GitHub [35], which might hinder our understanding of real human developers' activities. We detect 42 bot accounts, and they are excluded from this study. Then we resolve aliases using the automatic classification method proposed by Vasilescu *et al.* [36]. The reason for alias resolution is that developers may use different emails and usernames when making contributions, which causes contributions by the same individual to be linked to several accounts [37], [12]. 7 aliases are found and these accounts will be regarded as one.

In total, we build a cross-platform dataset of 37,060 Gitter chatters from 1,546,127 utterances and 89,858 GitHub developers from 395,664 activity records. Detailed statistics are shown in Table 1.

With the collected Gitter and GitHub data, we are able to extract 4,506 cross-platform contributors by mapping the usernames (a distinct and unique indicator of a user account) of GitHub contributors and Gitter chatroom participants of a specific community. Note that the mapping procedure is community-oriented, which means a CPC is a contributor on the GitHub repository as well as a chatter on the same chatroom of this community.

4 RQ1: CPCs' Role and Contribution

4.1 Methodology

First, we identify four types of development activity on GitHub as OSS contributions according to GitHub Docs [38] as follows:

- Commit: submit commits to the code repository.
- Report Issue: submit issue reports to track ideas, feedback, tasks, or bugs for improvement.
- Review Code: review to decide whether or not to approve the changes, or request further changes before the pull request is merged.
- Comment: post comments under the issue reports or pull requests.

Second, to identify the OSS roles of CPCs and their importance to the OSS communities, we build an OSS role taxonomy based on the *Onion Model* [19] and *Core-Periphery Model* [43], which is widely used in the OSS literature [16], [44], [45]. The role taxonomy includes seven roles: core

2. Onboarded contributors are those who have already onboarded on GitHub and made contributions. This will be discussed in detail in a latter section.

3. https://angular.io/

4. http://appium.io/

5. https://deeplearning4j.org/

6. https://www.docker.com/

7. https://ethereum.org/en/

8. https://nodejs.org/en/

9. https://www.typescriptlang.org/

10. When collecting data by sending requests, the response of API calls contains a field indicating user type, and if this field is "Bot", we consider this is a bot account.

TABLE 2: Classification of OSS roles

Priority	Role	How to determine
1	Core developers	Core developers are identified based on a Commit-based Heuristic following previous studies [20], [39], where core developers in a small number are deemed to account for 80% of the total commits of the project [40], [41], [42].
2	Peripheral developers	Similar to the identification of core developers, peripheral developers account for the rest 20% of total commits.
3	Issue reporters	Issue reporters are those who report issues.
4	Reviewers	Reviewers are those who review code.
5	Commenters	Commenters are those who send comments to issues and pull requests.
6	Readers	Repository forkers and dependent developers whose personal project is dependent on selected repositories are considered to be readers for they might read repository source code instead of simply using it.
7	Passive users	Passive Users just use the software as most of us use commercial software, and they are attracted to the OSS community due to its high quality and the potential of changes [19].

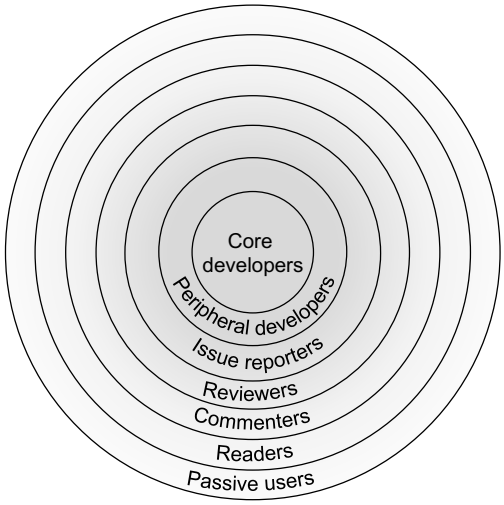


Fig. 2: Role-based OSS community structure

developers, peripheral developers, issue reporters, reviewers, readers, commenters, and passive users. Figure 2 shows the role-based OSS community structure. The onion-like structure reveals the different layers or levels of participation and engagement within the community. The inner layer a role is in, the more effort and contribution this role has devoted to the community. It also reflects the intersectionality-wise relationship between roles, *e.g.*, core developers in the center have an overlapping relationship with other roles, indicating core developers could report issues, review codes, and make comments. Table 2 describes roles' priority and determined rules. Note that, similar to the *Onion Model*, the role of core developers has the highest priority, and the role of passive users has the lowest priority. When a developer satisfies multiple roles, we only assign the role with the highest priority to the developer. This classification process is automatically accomplished according to each developer's activity record.

Third, we measure the social influence of CPCs in Gitter and GitHub networks respectively via Social Network Analysis (SNA) [46]. Social networks are utilized to model the process of information obtaining and spreading [47], and the propagation of information is often affected by the influence of certain nodes [48]. Therefore, we aim to unveil the social influence of CPCs to better understand their ability to disseminate knowledge in the network. We build social networks for Gitter and GitHub respectively for each community. The network of Gitter and GitHub share the same concept:

$$\begin{aligned} G &= \{V, E\} \\ V &= \{d_1, d_2, ..., d_n\} \\ E &= \{< d_i, d_j >\} \end{aligned} \tag{1}$$

where network G consists of a node set V and an edge set E. Node d_i represents each developer, and the edge $< d_i, d_j >$ denotes there is a certain relationship between node d_i and d_j.

When constructing the Gitter network, we define the relationship as "reply-to" in one dialog, following previous studies [2], [49]. The more times a developer communicates with different developers, the more nodes they will be connected to. When constructing the GitHub network, we define the relationship as "collaboration" in certain development activities, *i.e.*, one developer reviews another developer's code or commenting interaction on the same issue/pull request. The more times a developer collaborates with different developers, the more nodes they will be connected to. Both Gitter and GitHub networks are unweighted undirected networks.

Furthermore, to investigate CPCs' importance in coordinating communication and development, we visualize their influence levels based on the constructed Gitter and GitHub social networks by using Gephi [50]. Their influence levels are measured by the Semi-Local Centrality (SLC) [51], which is an efficient and prevalent approach to identify nodes that have a significant impact on network architecture and functionality, as shown in Equation 2:

$$\begin{aligned} Q(d_j) &= \sum_{d_w \in \Gamma(d_j)} N(d_w) \\ SLC(d_i) &= \sum_{d_w \in \Gamma(d_i)} Q(d_w) \end{aligned} \tag{2}$$

where $N(d_w)$ is the number of neighbors that can be reached within two steps from node d_w and $\Gamma(d_j)$ is the set of node d_j's neighbors. Then SLC value of node d_i can be calculated as the sum of the Q values of its neighbors.

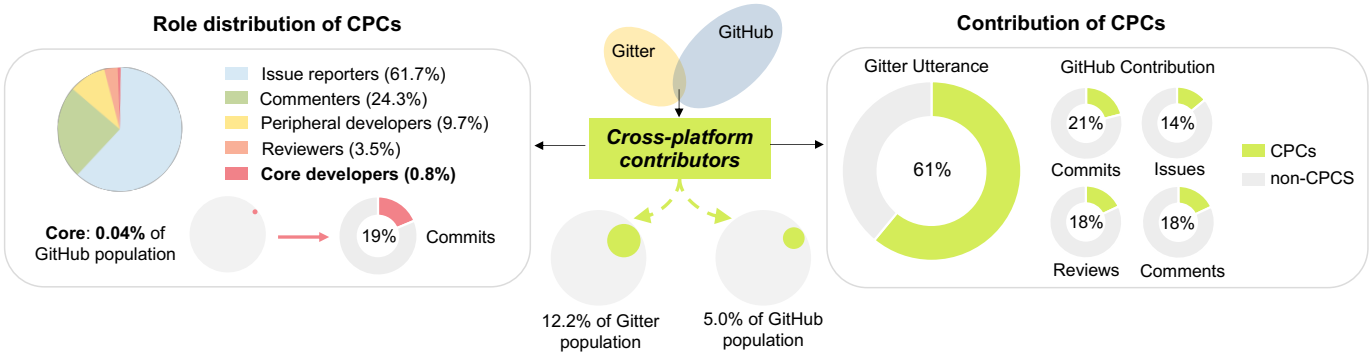
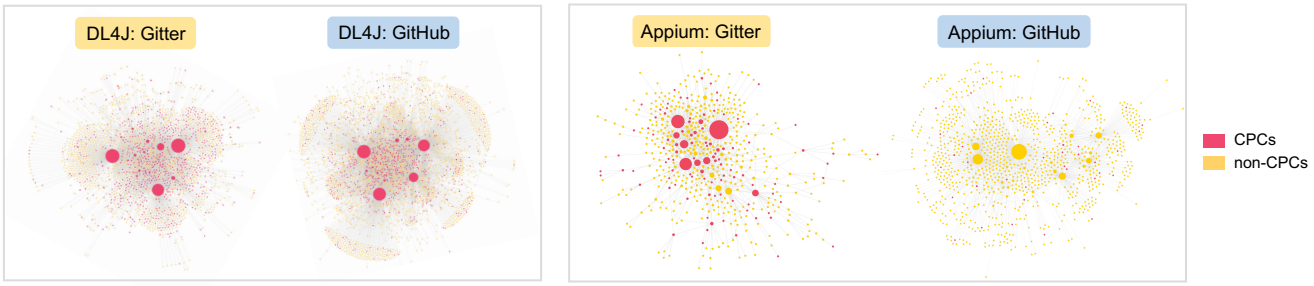


Fig. 3: Role analysis and contribution distribution of cross-platform contributors



(a) CPCs playing focal points in both communities (DL4J, Angular, Typescript)

(b) CPCs playing focal points only in Gitter, but not in GitHub community (Appium, Docker, Nodejs, Ethereum)

Fig. 4: Social network visualization of the Gitter and GitHub community of two examples. Node size indicates its SLC value.

4.2 Result and Analysis

Role and Contribution. Figure 3 shows the roles of CPCs and their contribution to OSS communication and development. (1) The left pie chart describes the distribution of CPCs' five roles. We can see that 61.7% CPCs are issue reporters, followed by commenters (24.3%), peripheral developers (9.7%), code reviewers (3.5%), and core developers (0.8%). 86% of them are issue reporters and commenters. The issue tracking function and pull requests on GitHub provide collaborators to record and follow the progress of every issue/pull request until it is resolved, which largely facilitates OSS management. However, GitHub might lack the live discussion of issues or pull requests [21]. In spite of the posting comments for issue/pull requests on GitHub or other forums, such discussion is in an asynchronous way, which lacks timely response and reduces efficiency. Therefore, issue reporters and commenters might prefer to participate more in Gitter live chat for instant issue/pull request communication. (2) The donut charts on the right in Figure 3 exhibit CPCs' contributions. We can see that, despite CPCs only accounting for 12.2% of the Gitter population, 61% of chat utterances are posted by them. The same phenomenon is observed in the GitHub platform. CPCs take up 5% of the GitHub population, but contribute to 21% commits, 14% issues, 18% reviews, and 18% comments. Even more exaggerated, the core developers in CPCs only account for 0.04% GitHub population, but contribute to 19% commits, as highlighted on the bottom left in Figure 3. This phenomenon complies with the Onion Model for healthy OSS communities in that, core developers are a small number of developers who make significant contributions to the open source systems [40], [41], [42]. These core developers with advanced knowledge and experienced skills have been communicating on Gitter and bringing their knowledge and skills by answering other developers' questions, which is a great benefit for Gitter participants who are seeking technical help.

Finding 1: CPCs only take up a small portion of developers on Gitter (12.2%) and GitHub (5.0%), but they contribute to more than 3/5 utterances on Gitter, and nearly 1/5 contributions on GitHub. Even more exaggerated, the core developers in CPCs only account for 0.04% GitHub population, but contribute to 19% commits.

Social Influence. Furthermore, to show their importance in coordinating communication and development, we visualize CPCs' SLC scores based on the Gitter and GitHub social networks. Each node denotes one developer, where CPCs are colored red, and other developers are colored yellow. The size of each node is associated with the SLC value of the developer, i.e., the bigger the node size is, the higher value of SLC it stands for. Based on the observation of the seven networks, we find that, in the social networks of Deeplearning4j, Angular, and Typescript, CPCs are focal points on both the Gitter network and the GitHub network. While in the social networks of Appium, Docker, Nodejs, and Ethereum, CPCs are focal points on the Gitter network but are non-focal points on the GitHub network. Figure 4 shows two examples for each of the two types. Among all the seven OSS communities, CPCs always play focal points on the Gitter platforms. In most cases (4/7), they are focal points only on the Gitter network, while in

some cases (3/7), they are focal points on both Gitter and GitHub. As introduced before, focal points in the network stand for larger SLC values and are more influential nodes compared to the rest, which means they tend to have a more significant impact on information dissemination and network architecture [51]. Therefore, the network visualization results indicate that CPCs are key to information spreading due to their outstanding performance in communicating with a great number of different developers. They might get used to communicating with other developers on Gitter every day, even though they are under heavy responsibility for contribution.

Finding 2: Among all the seven OSS communities, CPCs always coordinate OSS communications by playing as focal points on the Gitter platforms.

5 RQ2: CPCs' Communication Preference

5.1 Methodology

To explore what topics CPCs in different roles prefer, we adopt the topic taxonomy proposed in our prior work [2] which is extended based on Beyer *et al.*'s category of question categories on Stack Overflow [52], as shown in Table 3.

TABLE 3: Dialog topic taxonomy

Category	Subcategory	Topic
Domain-related	Solution-oriented	API Usage
		Review
	Problem-oriented	Unwanted Behavior
		Do Not Work
		Reliability Issue
		Performance Issue
		Test/Build Failure
		Error
		API Change
	Knowledge-oriented	Background Information
		New Features
		Design
		Learning
Non Domain-related		Social Chatting
		General Development

We randomly retrieve 150 CPC-participated dialogs from each community, leading to 1,050 dialogs with 1,493 CPC participants in total, then analyze the OSS roles and their participated topics orthogonally.

5.2 Results and Analysis

Topic. Table 4 illustrates the dialog topic distribution for different roles of CPCs. The colors are painted vertically to reflect the preference of different roles. The darker, the more times this role has participated in discussing this topic. We can see that: *core developers* prefer to participate in dialogs discussing "Design", which is an advanced topic in the high difficulty level [1]. This corresponds with the experienced programming skills and advanced project knowledge gained by the core developers. *Peripheral OSS developers* are more likely to participate in social chatting and discuss unwanted behaviors, errors, and performance issues. These observations also comply with the previous work [53], which reported that "Peripheral developers contribute irregularly and are typically involved in bug fixes or small enhancements." *Issue reporters* are nearly the largest population that participates in all the topics, and their percentages range from 29.6% to 66.7%. Those high percentages mean that issue reporters are actively involved in Gitter and discuss various topics. They are likely to benefit from the instant feature provided by the Gitter platform for receiving rapid responses when discussing API changes, performance issues, new features, something that does not work, etc. Moreover, among all the dialogs, they discuss API changes most, which might reflect the compatibility challenges faced by open-source software supply chain [54]. The *reviewers* are worthy of the name, they also prefer to participate in conversations about code review and reliability issues on Gitter. They showed little preference for "API change," "Unwanted behavior," and "Design." The *commenters* are not interested in API change (6.1%), however, they showed similar interest in other topics, ranging from 15.4% to 22.8%.

Finding 3: On the Gitter platform, core OSS developers are more interested in discussing designs, while peripheral OSS developers are more likely to participate in social chatting, discussing unwanted behaviors and errors.

TABLE 4: Conversation topics of different roles of CPCs

Topic			Core	Peripheral	Issue	Reviewer	Commenter
Domain-related	Solution-oriented	API Usage	13.6%	10.6%	41.1%	16.9%	17.8%
		Review	10.4%	7.3%	39.6%	25.0%	17.7%
	Problem-oriented	Unwanted Behavior	19.8%	13.5%	39.6%	7.2%	19.8%
		Do Not Work	8.7%	7.2%	46.4%	17.4%	20.3%
		Reliability Issue	14.8%	7.4%	29.6%	25.9%	22.2%
		Performance Issue	9.7%	11.3%	51.6%	11.3%	16.1%
		Test/Build Failure	16.7%	8.3%	41.7%	12.5%	20.8%
		Error	20.7%	12.8%	32.9%	12.2%	21.3%
		API Change	18.2%	6.1%	66.7%	3.0%	6.1%
	Knowledge-oriented	Background Infor	17.8%	10.4%	40.0%	11.9%	20.0%
		New Features	9.1%	4.5%	50.0%	20.5%	15.9%
		Design	32.0%	0.0%	40.0%	8.0%	20.0%
		Learning	15.8%	5.9%	35.6%	19.8%	22.8%
Non Domain-related		Social Chatting	15.4%	15.4%	33.8%	20.0%	15.4%
		General Development	6.2%	9.2%	46.2%	23.1%	15.4%

6 RQ3: CPCs' Behavioral Consistency

6.1 Methodology

In order to examine whether CPCs behave similarly or differently when switching platforms, we investigate the behavioral consistency of CPCs across two platforms in terms of activeness, influence, and collaboration.

Activeness Consistency. We define the activeness for individual developer d_i on Gitter and GitHub as Equation 3 and 4. The utterances of a developer in the chatroom indicate his/her activeness on Gitter. The GitHub activeness is correlated to the four types of development activity introduced before, *i.e.*, commit, issue, review, and comment. Since the variability among these values is huge, we apply normalization to Gitter and GitHub activeness, notated with a subscript nor. As shown in Equation 3 and 4, the Gitter activeness is normalized straight away, while the GitHub activeness is measured by an equal-weighted summation of the normalized count of the four kinds of activity mentioned before. In this way, both Gitter Activeness and GitHub activeness are scaled into a range from 0 to 1. Then correlation analysis is employed to investigate consistency. Since the activeness data does not meet the requirement of bivariate normality, we apply Spearman's correlation analysis [55] to examine whether CPCs' Gitter activeness and GitHub activeness are consistent.

$$\begin{aligned} Utterance(d_i) &= \sum utterance(d_i) \\ Activeness\_Gitter(d_i) &= Utterance_{nor}(d_i) \end{aligned} \tag{3}$$

$$\begin{aligned} Commit(d_i) &= \sum commit(d_i) \\ Issue(d_i) &= \sum issue(d_i) \\ Review(d_i) &= \sum review(d_i) \\ Comment(d_i) &= \sum comment(d_i) \\ Activeness\_GitHub(d_i) &= \frac{1}{4} Commit_{nor}(d_i) \\ &\quad + \frac{1}{4} Issue_{nor}(d_i) \\ &\quad + \frac{1}{4} Review_{nor}(d_i) \\ &\quad + \frac{1}{4} Comment_{nor}(d_i) \end{aligned} \tag{4}$$

Activeness consistency is investigated from a macro perspective and a dynamic perspective. In macro activeness consistency analysis, we investigate the correlation of all CPCs in all seven communities from the beginning of usage of both Gitter and GitHub until the ending date of our data collection. In the dynamic analysis, we measure the cumulative correlation change of each project over time. The correlation coefficient is iteratively recalculated every incremental 30 days to observe changing trends.

Influence Consistency. We define four levels of influence according to quartiles of SLC values of the Gitter network and GitHub network. Respectively, Gitter-Q4 denotes the quarter of the highest SLC in Gitter, Gitter-Q3 denotes the SLC of this quarter ranging from the median to the third quartile, and so on. By tracing the influence migration flow between Gitter and GitHub, we can observe their influence consistency.

Collaboration Consistency. To investigate collaboration consistency, we analyze the overlap between communication groups on Gitter and collaboration groups on GitHub in two steps. First, we cluster the CPCs from the two platforms into communication or collaboration groups respectively. For the Gitter platform, we consider the developers who have communicated within the same dialog as a communication group. For the GitHub platform, we consider the developers engaged in the same development activity as a collaboration group. The goal is to investigate CPCs' collaboration consistency, therefore, we retain groups that contain at least two members. Second, we pair the two groups taken from the two platforms and calculate their degree of overlap. The extent of overlapping is calculated via Jaccard coefficient. For example, 100% overlap denotes that the developers in the two groups are the same. 50% overlap denotes that half the developers in the two groups are the same. If the degree of overlap between groups on Gitter and groups on GitHub is high, we consider the CPCs' collaboration is consistent across the two platforms, and vice versa.

6.2 Result and Analysis

To investigate CPCs' behavioral consistency, we compare their behaviors between two platforms in terms of activeness consistency, influence consistency, and collaboration consistency.

Activeness Consistency. Figure 5 shows the macro correlation analysis result of Gitter and GitHub activeness with the Spearman's correlation coefficient r equals to 0.17 (p-value $= 4.99 \times 10^{-30}$). It means there is only a weak and slight correlation between Gitter and GitHub activeness. The majority of developers cluster near the horizontal and vertical axis instead of the center, indicating their $Activeness\_Gitter$ does not align with $Activeness\_GitHub$ in most cases. Even though CPCs as a whole show active participation on both Gitter and GitHub platforms according to Finding 1 (CPCs account for more than 3/5 of the Gitter utterances and nearly 1/5 of GitHub contributions), the little correlation between $Activeness\_Gitter$ and $Activeness\_GitHub$ implies that the consistent activeness does not exist anymore individual-wisely. The far-out values that are highly active on one platform while much less active on the other might be because the developers' effort is limited so they can not afford to be both highly active on Gitter and GitHub.

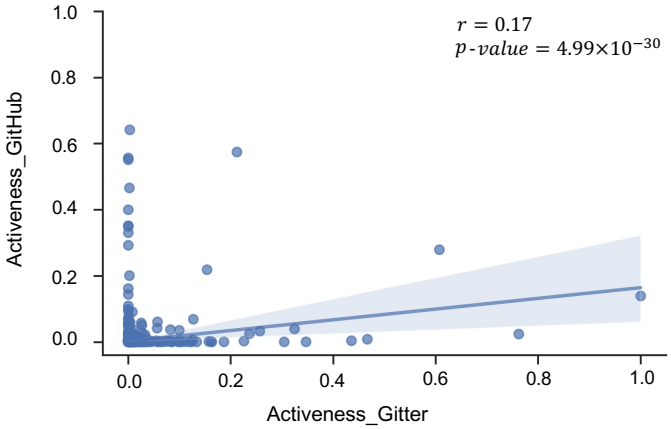


Fig. 5: Macro activeness correlation of cross-platform contributors.

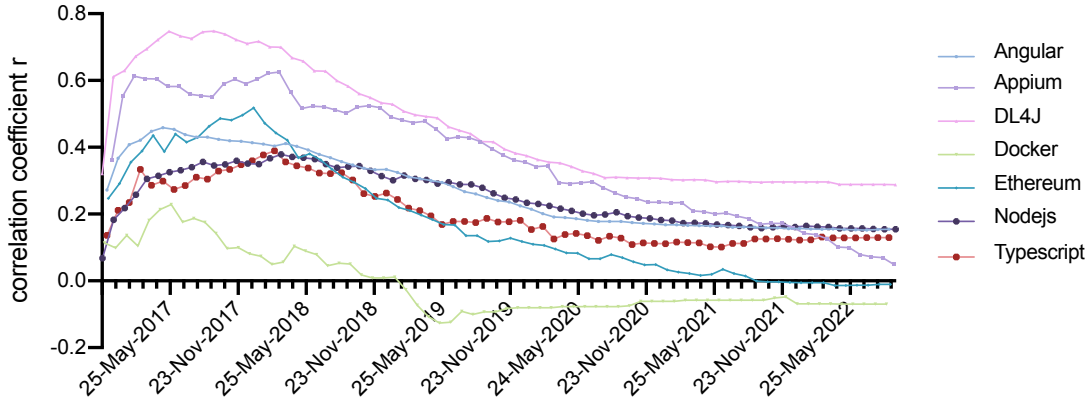


Fig. 6: Cumulative activeness correlation change

Figure 6 visualizes the change of culmulative activeness correlation throughout time. In all cases, the correlation coefficient r rises to a peak during the early stage of Gitter usage. Projects like DL4J and Appium's peak values can even indicate a strong and moderate correlation, respectively. Then r starts to decline until reaching a flattening out, where correlation is weak in the late and stable stage of OSS. This is probably related to the evolution of the OSS community. When Gitter was first introduced to these projects, the project contributors were actively developing and testing on GitHub and discussing these developmental problems on Gitter, resulting in a relevantly high correlation during this period. However, with the evolution and growth of the OSS communities, more commenters and issue reporters were engaged in Gitter live chat, asking questions such as API usage that are less relevant to GitHub developmental activities and making occasional contributions [44]. Additionally, the active contributor might retire or take breaks from making GitHub contributions [20] but are still active on Gitter.[11] These CPCs' discussions are active on Gitter but their contribution is much less on GitHub, leading to a mismatch and thereby a weakening correlation relationship.

Finding 4: Broadly, there is merely a weak correlation between Gitter activeness and GitHub activeness. However, the cumulative activeness correlation changes over time, following a fast rise and a steady decline trend, until converging on a weak correlation.

Influence Consistency. To exhibit the flow of CPCs' influence on GitHub and Gitter, we draw the Sankey diagram as shown in Figure 7. The length of black lines represents the number of CPCs that fall into the corresponding SLC quarter. The width of the connections is proportional to the number of CPCs who shift from a quarter in one platform to a quarter in the other. If the connections are straight, the influence levels remain the same. If the connections are curved, the influence levels change. We can see that about 70.2% connections are curved, which means most CPCs' social influence largely migrates between platforms. The results indicate that CPCs with lower influence are able to achieve higher social influence on the other platform and

11. This phenomenon is observed in RQ4 (Section 7.2.3) in our study.

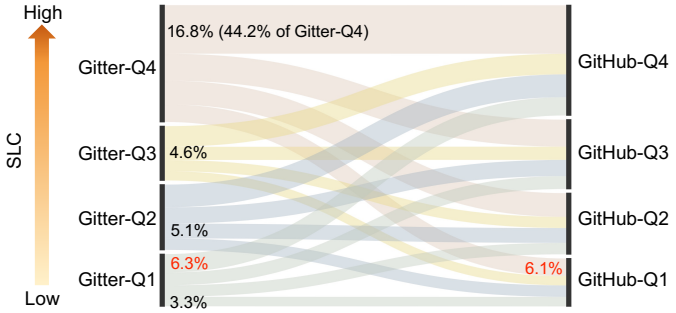


Fig. 7: CPC's influence migration between Gitter and GitHub

vice versa, as notated in red in Figure 7. In the meanwhile, as annotated in black, only 29.8% connections are straight, which means only these CPCs retain their degree of influence at the same level. It is noticeable that a thicker flow lies between Gitter-Q4 and GitHub-Q4, showing that nearly half (44.2%) of Gitter-Q4 CPCs retain their high influence while communicating on the Gitter. These CPCs are typically OSS core developers or leaders who are involved with the OSS communities for a relatively long period and are responsible for guiding and coordinating communication and development [19], [39], [40].

Finding 5: The level of CPCs' social influence frequently changes between Gitter and GitHub. Despite Some contributors have a low social influence level on GitHub, they can be highly influential on Gitter.

Collaboration Consistency. Table 5 shows the number of identified Gitter communication groups and GitHub collaboration groups, and the average size of the groups. The '100% overlap' columns show the percentage of GitHub groups that are 100% overlapping with the groups on Gitter, and the percentage of Gitter groups that are 100% overlapping with the groups on GitHub. The '50% overlap' columns show the percentage on the 50% overlapping. We can see that except for Deeplearning4j, most of the studied OSS communities have zero or small overlaps between their Gitter groups and GitHub groups, indicating CPCs' collaboration groups on Gitter rarely intersect with their collaboration groups on GitHub.

TABLE 5: Overlaps between GitHub collaborative groups and Gitter chat groups

Community	CPC in GitHub		CPC in Gitter		100% overlap		50% overlap	
	# Group	# Dev per group	# Group	# Dev per group	% GitHub	% Gitter	% GitHub	% Gitter
Angular	1,637	2.5	6,951	2.7	2.9%	0.7%	12.3%	12.0%
Appium	0	0.0	139	2.3	0.0%	0.0%	0.0%	0.0%
Dl4j	1,371	2.7	3,126	2.7	26.0%	11.4%	73.2%	57.0%
Docker	0	0.0	35	2.1	0.0%	0.0%	0.0%	0.0%
Ethereum	5	2.0	21	2.0	0.0%	0.0%	0.0%	0.0%
Nodejs	7	2.1	72	2.0	0.0%	0.0%	0.0%	0.0%
Typescript	631	2.4	2,149	2.7	1.6%	0.5%	8.9%	9.4%

This reveals that the developers who collaborate with each other on GitHub do not frequently communicate on Gitter, and vice versa. This may be because the two platforms serve different concerns for developers. For example, the group of developers who have collaboratively resolved issues on GitHub, are likely not to discuss them again on Gitter.

Finding 6: The CPCs' collaboration groups on Gitter rarely intersect with their collaboration groups on GitHub.

7 RQ4: Gitter's Impact on Contribution

7.1 Methodology

To understand Gitter's impacts on GitHub contribution, we investigate its impact on new contributors, onboarded contributors as well as returned code contributors, respectively.

Impact on New Contributors. Aiming to explore whether live chat facilitates new contributor attraction, We analyze those new GitHub contributors who are attracted via Gitter and how they are attracted. First, we identify the new contributors that are likely attracted via Gitter, based on whether they have been active in Gitter before their first contribution to GitHub. Second, we collect the 'last' conversations they were involved in Gitter before their first contribution as the attracting conversations. Third, we employ an *open card sort* process [56] to manually categorize the attracting conversations based on their contents. We enroll a team consisting of three Ph.D. students. All of them are fluent in English and have done either intensive research work with software development or have been actively contributing to open-source projects. The open card sort starts with no predefined categories, then the team members individually assign categories to the same dialogs. The process is conducted in multiple rounds. In the first round, all participants label 10% sampled dialogs with a thorough discussion to obtain conceptual coherence. A shared pool of categories is utilized and carefully maintained, and each participant could select existing categories from and/or add new category names to the shared pool. The sorting process ends when there is no new category added for two consecutive rounds. The average Cohen's Kappa is 0.84, which indicates substantial agreement.

Impact on Onboarded Contributors. Onboarded contributors refer to contributors who have onboarded on GitHub platforms and made contributions. To find out Gitter live chat's impact on onboarded contributors' contributions, we categorize GitHub developers into two groups based on the usage of Gitter.

- G1: GitHub developers who use Gitter, *i.e.*, CPCs.
- G2: GitHub developers who do not use Gitter, *i.e.*, non-CPCs on GitHub.

We compare the differences between the two groups in terms of the number of commits, issue reports, reviews, and comments. Note that this comparison between the two groups is role-wise. As introduced in Section 4.1, the role taxonomy is onion-structured, which means that the centered roles in high priorities could participate in lower-priority activities. For instance, peripheral developers could also report issues and make comments. Therefore, as shown in Figure 8, we compare the contributions of CPCs and non-CPCs of the specific roles that would make certain types of the above-mentioned contributions. For example, for commit contribution, the comparison subject G1 contains CPCs that made commits (*i.e.* core and peripheral developers in CPCs), and G2 includes non-CPCs that made commits (*i.e.* core and peripheral developers in non-CPCs). And for issue report contribution, the object roles are core developers, peripheral developers, and issue reporters.

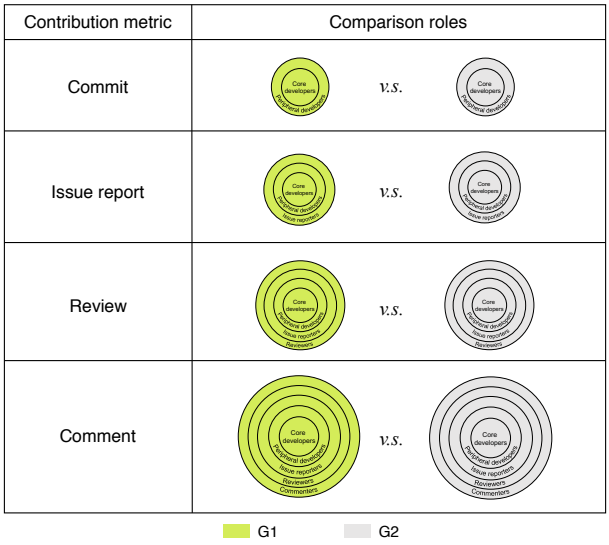


Fig. 8: Role-wise comparison

In addition, to achieve a fine-grained understanding of their GitHub contribution, we also compare the following aspects in a role-wise way:

- Response rate: the rate of issues/PRs proposed by a specific developer getting responses from others.
- Issue closure rate: the rate of issues opened by a specific developer getting closed.

- PR approval rate: the rate of pull requests proposed by a specific developer getting approved.
- Issue resolution time: the time interval between an issue's opening and closing (in seconds).
- PR approval time: the time interval between a PR being proposed and approved (in seconds).
- Commit frequency: the number of code commits made by a specific developer per day.

Hypothesis testing is used to validate if the differences are significant, with the null hypothesis to be:

H_0: *The CPCs' contributions are smaller than or equal to ($\leq$) that of developers who do not chat on Gitter.*

Note that hypothesis testing is applied on each metric, so there will be a detailed hypothesis for each metric, with the contribution type displayed in the description of H_0. Metrics that are not related to time can be measured by the accumulated number. However, if the metric is issue resolution time or PR approval time, the shorter time indicates better performance, so the operator in H_0 should be $\geq$.

Impact on returned code contributors. Prior research on the life cycle of contributors revealed that code contributors might disengage in making contributions for a period of time and some of them would return to OSS after taking a break [20]. Since code contributors are the backbone of the OSS product, the loss of them is a major problem in OSS communities and ecosystems [57] that might lead to community disruption, productivity decline [58], and product quality reduction [59]. Therefore, we are motivated to explore the impact of live chat on code contributors' returning. To that end, we first investigate whether contributors participate in Gitter live chat during inactive periods, then we calculate the return probability of contributors who chat during inactivity and those who do not, respectively. The detailed steps are as follows:

(1) We define three states that can reflect OSS contributors' turnover according to previous work [20] as follows:

- ACTIVE: a period during which the contributor contributes to GitHub actively by committing.
- BREAK: longer-than-usual pauses between two ACTIVE states. Note that the threshold to determine whether a pause period is longer than usual is set based on each developer's own work rhythm. Given a pause sequence $P =< p_1, p_2, ...p_n >$, the algorithm first utilizes the *far out values* approach [60] to define a developer-specific threshold $T_{fov} = Q_3(P) + 3 \times IQR$, where Q_n denotes the n^{th} quartile of P, and $IQR = Q_3(P) - Q_1(P)$ the interquartile range (or *H-spread*). As contributors' work rhythms are likely to change over time, to achieve naturalness, the algorithm introduces *sliding window* mechanism. The BREAK periods are defined as the longer-than-usual pauses that are above T_{fov} within the window.
- GONE: If a contributor has been in BREAK state for more than one year, he/she will be considered as GONE instead.

(2) We can generate the state sequences $Life(d_i)$ for individual developer d_i according to his/her activities on GitHub over time. E.g., $Life(Tom) = \{ACTIVE, BREAK, ACTIVE, GONE, ACTIVE\}$

(3) We consider a 'Return' happens if the state changes from BREAK to ACTIVE, or GONE to ACTIVE. Indeed, the state sequence $Life$ is a Markov chain [61], in which the probability of each state depends only on the previous state. Thus, given the state sequence $Life = \{x_1, x_2, ...x_n\}$, we can calculate the return probability as the state transition probability:

$$\begin{aligned} P(Return) =& P(x_i = ACTIVE | x_{i-1} = BREAK) \\ &+ P(x_i = ACTIVE | x_{i-1} = GONE) \end{aligned} \quad (5)$$

where x_i denotes a certain state in $Life$.

7.2 Result and Analysis

In this subsection, we will introduce Gitter's impact on OSS contribution in three aspects: new contributors, onboarded contributors, and returned contributors.

7.2.1 Impact on New Contributors

Among all the GitHub developers, we find that 2.5% (2,204/89,858) developers have participated in Gitter dialogs before their first contribution to GitHub.

Fang *et. al* [62] recently reported that tweets containing links to GitHub repositories cause an around 2% increase in committers. In this study, we observe a higher increase in attracting new contributors via Gitter. The higher increase may be due to the difference between the two platforms in that, Twitter is a communication platform for socialization among any individual who registered, while Gitter is a communication platform for technical discussions among OSS developers.

To analyze how Gitter attracts new contributors, we collect 2,204 dialogs that they last participated in before their first contribution as attracting conversations. Limited by the effort, we obtained a representative sample (328 dialogs) with 95% confidence level and 5% confidence interval, carefully read their utterances, and classify them into six categories as shown in Figure 9. In terms of motivation, 89% new contributors are self-motivated (NC1-4, NC6), and 7.3% are motivated by core developers. In terms of identity, 56.7% (NC1, NC3, NC6) are OSS product users, and 39.6% (NC2, NC4-5) are other developers who are interested in the OSS products. Note that, 3.7% are rare cases that are considered as others. The explanations and examples of NC1-NC6 are as follows.

NC1. OSS product users who seek help for using the OSS product (35.1%). 35.1% attracting conversations can indicate that the new contributors are the product users who use the product as a third-party library in their own code. They actively seek help about usage problems from the community, *e.g.*, "How can I implement video streaming using NodeJS?".

NC2. Supportive developers who are keen on sharing knowledge, new techniques, and experience. (24.1%) 24.1% attracting conversations can indicate that the new contributors are supportive and sociable. They tend to share their knowledge or experience and discuss new techniques with other developers. *e.g.*, they answer other developers' questions, participate in complex technical discussions, provide

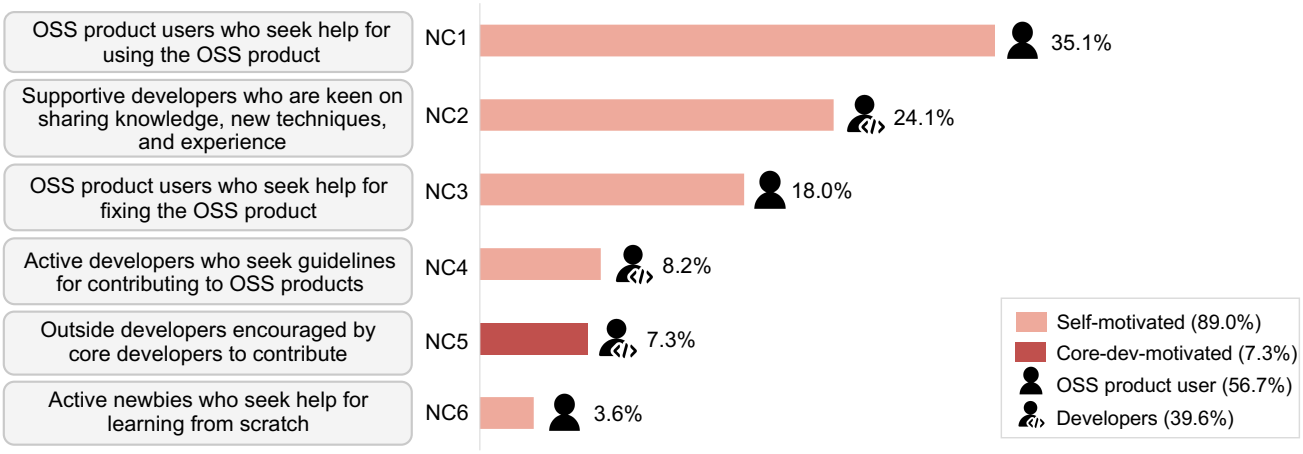


Fig. 9: The identity and motivation of new contributors imported via Gitter.

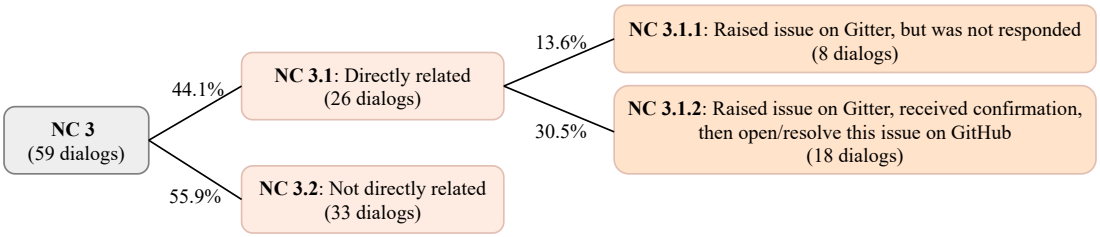


Fig. 10: Cross-platform connection investigation results of NC3 dialogs.

external links for learners, and disseminate newly released tools or technologies.

NC3. OSS product users who seek help for fixing the OSS product. (18%) 18% attracting conversations can indicate that the new contributors are the OSS product users who seek help for fixing reported issues that may annoy them. *E.g.*, "In order to solve issue #30507, I'm looking into this link, which should get the whole possible type instead of a concrete type. Are there util methods to replace type parameters in an expression?"

To further assess Gitter's impact on attracting issue reporters and resolvers, we investigate if a connection exists between the issue raised in the Gitter chatroom and NC3-CPCs' first contribution. As shown in Figure 10, 44.1% of issues raised in NC3 dialogs are directly relevant to the GitHub contribution (reporting the issue/committing code to resolve the issue) made by the CPC under investigation. Among them, 13.6% of developers were not responded so they chose to report/resolve it on their own, and 30.5% of developers received confirmation about the issue they raised on Gitter and reported/resolved it on GitHub. This indicates that raising problems on live chat encourages passive users and readers to take their first step towards GitHub contribution, reporting and fixing OSS defects to improve OSS quality.

NC4. Active developers who seek guidelines for contributing to OSS products. (8.2%) 8.2% attracting conversations can indicate that the new contributors are active developers who show explicit interest in contributing but don't know How-To, *e.g.*, "Does anyone know how to submit a PR to this repo?"

NC5. Outside developers encouraged by core developers to contribute. (7.3%) 7.3% of sampled new contributors are outside developers who are encouraged by core developers, and then make their first contribution to the GitHub repository. We will discuss more about this in section 8.

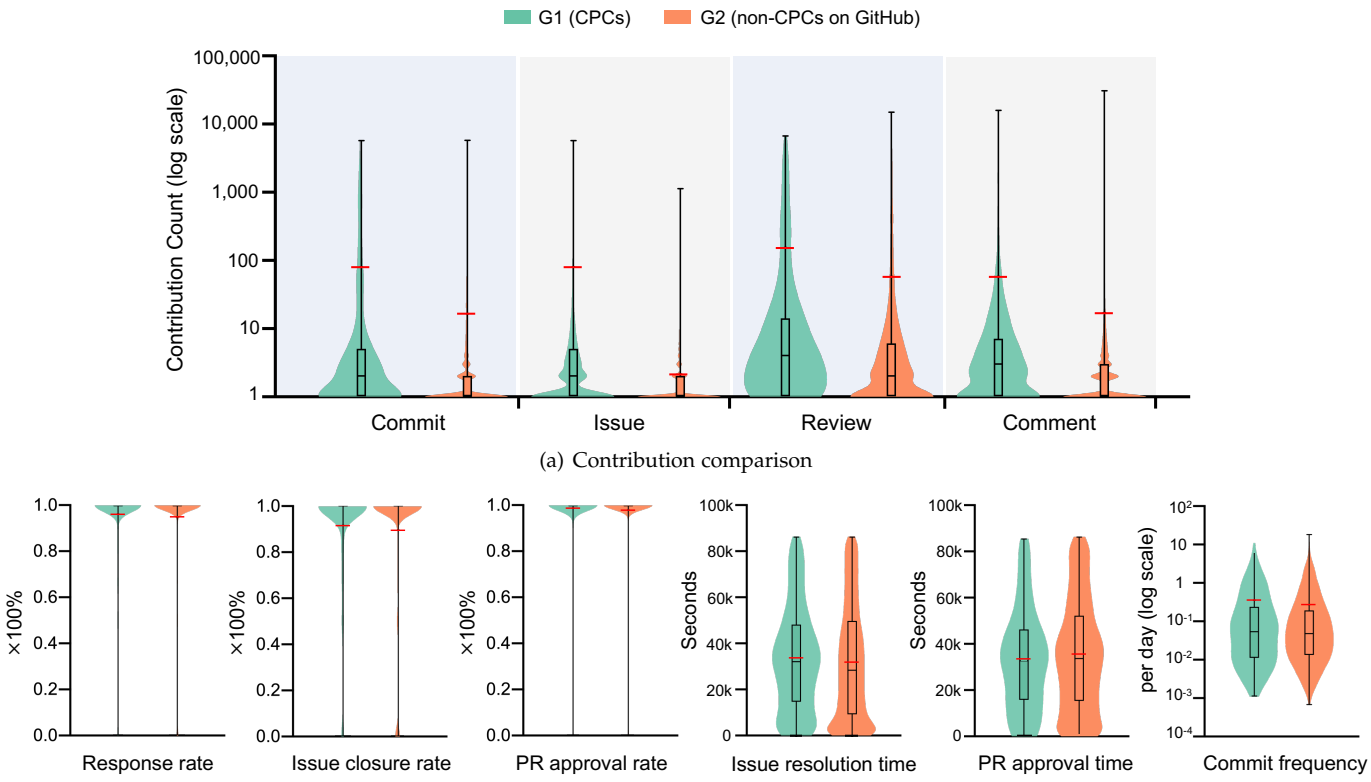
NC6. Active newbies who seek help for learning from scratch. (3.6%) 3.6% of new contributors are active newbies who seek help for learning from scratch. *e.g.*, "I am a student who wants to make a private blockchain using Ethereum. But I do not know what to start with so I'm about to start. I'd like some advice on what I should do first." They treat Gitter as their first landing point of OSS development, learning via Gitter, and then switching to GitHub contributors.

Finding 7: As a recently released communication platform, the amount of new contributors who participated in live chat before first GitHub contribution accounts for 2.5% of the population. 56.7% of them are OSS product users, and 39.6% are other interested developers. Gitter plays a positive role in promoting issue reporting and resolving. Core developers are one of the motivating factors that can attract new contributors via Gitter.

7.2.2 *Impact on Onboarded Contributors*

To compare the CPCs' onboarded GitHub contribution with that of other GitHub developers, we draw the violin plots in Figure 11. The inside box plots show the quartiles and median of the distribution, and the horizontal red line denotes the mean value. To test the significance of the difference, we use the Wilcoxon Rank Sum Test, a non-parametric alternative to the two-sample t-test [63], because none of the contribution data follows Normal distribution.

In Figure 11(a), we can see that the contribution from CPCs outstrips non-CPCs in terms of all four types of devel-



(a) Contribution comparison

(b) Fine-grained activity comparison

Fig. 11: Onboarded contribution comparison between OSS developers who use Gitter (CPCs) and those who do not (non-CPCs)

opment activities. The middle and upper part of the violin of CPCs is wider than non-CPCs, indicating CPCs include more active contributors. A number of non-CPCs cluster at the bottom of the violin makes the upper part less wide, which means there are more one-time contributors in developers who do not chat on Gitter. We conduct the single-sided test on the alternative hypothesis "The accumulated number of CPCs' contributions is larger than that of developers who do not chat on Gitter", from which we come to a conclusion that the alternative hypothesis is accepted with all p-value <0.001, indicating a significant difference between the CPCs' contribution and others' contribution.

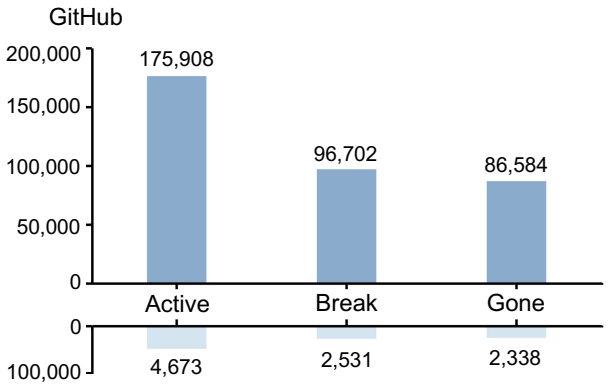
However, according to Figure 11(b), the fine-grained activity difference between CPCs and non-CPCs is not as significant as that of the four types of contributions. According to the hypothesis testing result, only the p-value for *PR approval time* is less than 0.001, indicating CPC-proposed pull requests take a shorter time to be resolved than those proposed by non-CPCs. While p-values for other activities are all larger than 0.1. From the violin plots, we can see that the data distribution of *Response rate*, *Issue closure rate*, *PR approval rate* and *Commit frequency* of CPC and non-CPCs is quite similar, while CPC's *Issue resolution time* are slightly longer than that of non-CPCs.

The communication on Gitter might enable developers to learn more knowledge and skills about the OSS product, as well as to obtain closer access to repository core members [64] which might facilitate later GitHub collaborations with them, such as making joint commits, encouraging newcomers to report issues they run into, giving impetus to the progress to pull requests and code reviewing. A similar phenomenon has been observed on Twitter by Fang *et al.* [65], that newly attracted GitHub contributors tend to collaborate actively with developers with whom they had prior Twitter interaction. But when it comes to fine-grained activities, the advantage of CPCs is not as significant as before. This might be because issue/PR-related metrics (*e.g.*, *Issue closure rate*) are affected by not only the developers who initiated the issue/PR, but also by the difficulty of the problem itself. For instance, Sahar *et al.* [27] reported that issues discussed in Gitter chat take longer time to be resolved than those not mentioned in Gitter chat for a possible reason that discussed issues are so difficult that developers need to raise attention in chatrooms.

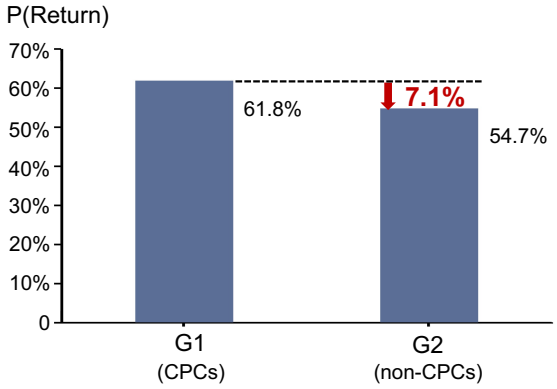
Finding 8: The communication on Gitter might have a positive impact on some GitHub onboarded contributions, since GitHub developers who communicate on Gitter have a significantly higher contribution with regard to commit, issue, review, comment and PR approval time than those who do not communicate on Gitter.

7.2.3 Impact on Returned Code Contributors

Figure 12(a) shows two quadrants representing GitHub and Gitter, respectively. The upper quadrant shows the number of developers (*i.e.* code contributors) in the corresponding GitHub states, and the lower quadrant tells us how many of them communicate in the Gitter platform when they are in the corresponding state. Note that the sum of the Gitter quadrant is larger than the population of CPCs because a



(a) OSS developer life-cycle across Gitter and GitHub



(b) Comparison on return probability

Fig. 12: Statistics of Gitter's impact on returned code contribution

sequence of a developer may contain several states. The same reason applies to the GitHub quadrant as well. In spite of being inactive (BREAK or GONE) on GitHub, about 2.7% of developers participate in live chat in the meantime. This indicates that not making any contribution to the GitHub repository does not mean the contributors leave the community. They might be active on other platforms that are related to this project, such as communicating on live chat. Inspired by this phenomenon, we further investigate whether developers who communicate on live chat have a higher possibility of return. As shown in Figure 12(b), G1 (developers who use Gitter) has a higher probability of returning to ACTIVE than G2 (developers who do not use Gitter) by 7.1%, indicating live chat communication has a positive effect on inactive developers' returning.

Finding 9: The communication on Gitter has a positive impact on returned GitHub contributions. GitHub Developers who communicate on Gitter have a higher probability of returning to contribute than those who do not communicate on Gitter. We observe that there are 2.7% developers who have been inactive on GitHub still participate in discussions on Gitter.

8 Implications

In this section, we provide implications and suggestions for individual developers, OSS communities, and OSS researchers.

8.1 Individual Developers

Live chat is a good landing point for joining the OSS communities (Finding 1, Finding 4, Finding 7, Finding 8). For new developers who would like to join the OSS communities to contribute, live chat is a good landing point since 12.2% of Gitter chatters are cross-platform contributors, and they contribute to more than 3/5 Gitter utterances and nearly 1/5 of GitHub contributions. These CPCs are experienced and important contributors in the OSS communities, it is a good ice-breaking activity once the new developers and the experienced ones have gotten to know each other. If new developers could participate in discussions with them, they will have the opportunity to seek help for using the OSS product, be taught knowledge and experience, be encouraged by core developers, etc. on the Gitter platform.

Leverage live chat to raise awareness of GitHub issues (Finding 1, Finding 3, Finding 6, Finding 7). According to the analysis of RQ1 and RQ2, issue reporters are the majority of CPCs, and they are actively engaged in various topics. Live communication on Gitter enables developers to gain closer access to these issue reporters and possibly receive guidance from them. In Figure 9, 18% of OSS product users are seeking help for fixing issues about OSS products before their first contribution. This might be because live chat platforms such as Gitter provide an opportunity for instant communication with other issue reporters in a more timely way. Besides discussing discovered bugs and resolution ideas, developers can also share similar experiences and inquire about the current progress of an issue. Therefore, if developers encounter issues that are out of their ability or lack attention on GitHub, they can resort to live chat, to bring it to other experienced developers that are supportive to help.

8.2 OSS Communities

It's not a "Good-bye", it's a "See you in Gitter" (Finding 9). Retaining onboard contributors has always been an important goal for any OSS community [64]. We found that the developers who use Gitter have a higher probability to return, than the developers who do not use Gitter (Figure 12(b)). Therefore, to maintain a sustainable lifecycle of OSS developers, and prevent developers from dropping the community, the communication platform could help. OSS community could closely cooperate with communication platforms, providing support or integration mechanisms that the OSS contributors can use handily. OSS stakeholders should pay more attention to the maintenance of their communication community, *e.g.*, (1) promote official chat platforms on the website and GitHub repository *README* file, encouraging developers and users to participate in it; (2) host online technical salons, events, or Q&A sessions to encourage regular engagement, keep the community active, and foster collaboration among members; (3) consider hiring moderators or community managers to address inappropriate behaviors that undermine the positive atmosphere such as irrelevant advertising.

Live chat is an important place to absorb new contributors (Finding 7). Absorbing new developers to contribute

is another important goal for OSS communities [64]. In RQ4, we found that Gitter is an important place to absorb new contributors, and these new contributors can be summarized into different categories in terms of motivation and identity. We notice that core developers could play an essential role in encouraging newcomers. To better exhibit how they successfully encouraged newcomers, we revisit the dialogs in RQ4 which belong to NC5, and choose one conversation to exemplify, as shown in Figure 13. We can see that at first newcomer *D1* discovered a bug and reported it in the Gitter chatroom to raise attention. Then the core developer *D2* encouraged the newcomer twice as well as provided detailed guidelines to the newcomer, and successfully motivated their first contribution. Being a core developer implies not only successful project management but also a high level of influence on others. It is a good practice that more OSS core developers can interact with newcomers or periphery developers via live communication platforms to encourage more contribution. It is also suggested that OSS stakeholders not only provide accessible onboarding guidelines on code-hosting platforms but also their communication platforms.

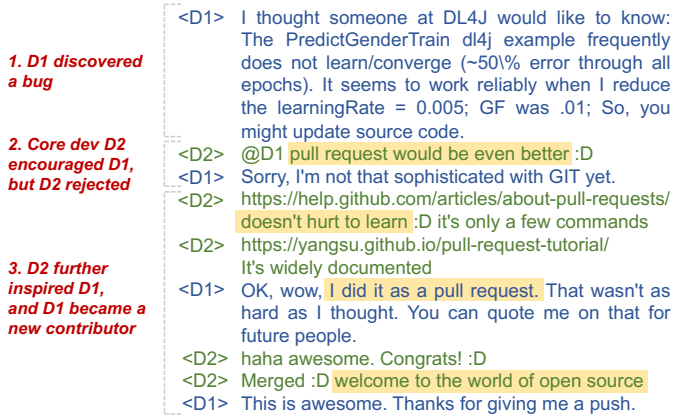


Fig. 13: A Gitter conversation about how core developers encourage developers to make contribution

Benefit from the knowledge sharing among multi-platforms (Finding 1, Finding2, Finding 5). In RQ1 and RQ3, we find that, although the population of CPCs is small, they are responsible for a larger quantity of contribution and a higher influence, and CPCs undertake the burden of coordinating communication by playing as focal points on Gitter. One possible reason might be that these developers have a wider way of absorbing new knowledge, techniques, and experience, and can disseminate them across multiple platforms. Therefore, to benefit from knowledge sharing among multi-platforms, OSS communities are suggested to pay more attention to building multi-platforms to enrich their OSS ecosystem.

8.3 Researchers

Revisit the lifecycle of OSS and its contributors from a multi-platform perspective (Finding 9). Most existing studies investigate the lifecycle or contributions of OSS developers from the most popular platform, i.e., GitHub. In RQ4, we notice that although some developers are defined as BREAK or GONE from the GitHub platform, they still keep active on others, and make other forms of contribution to the OSS communities. Therefore, it is worth revisiting the lifecycle of OSS and its contributors from a multi-platform perspective, as well as exploring what other forms of contribution they make.

Explore how OSS developers collaboratively resolve issues or review code on live chat (Finding 3 and Finding 7). In RQ2 and RQ4, we find that OSS developers might launch technical conversations that discuss complex issues or code reviews on Gitter, taking its advantage of the timely response and closely linking to GitHub. By analyzing these complex technical conversations, we could better understand the OSS collaboration mechanism. Sahar *et al.* [27] took a first attempt at analyzing the duration of issue resolution via Gitter. But there is still a lot worthy of exploring, such as how they review code via live chat and whether discussing issues via live chat helps increase the quality.

Automatically answer the questions on live chat with high confidence, especially for developers who seek help for joining the OSS community (Finding 7). In RQ4 (Figure 9), 8.2% of sampled developers seek instructions or guidelines to follow to contribute. Similar questions might have been answered before, however, if not answered correctly or timely, the OSS community might risk losing potential new contributors. If there is a Q&A bot that can automatically answer those questions with high confidence, the OSS community will benefit from it in terms of attracting newcomers and improving communication efficiency.

9 Threats to Validity

This section discusses the threats to validity of our work.

External Validity. The external threats are related to the generalizability of the proposed approach. Our empirical study used seven most participated open source communities from Gitter. Although we generally believe all communities may benefit from knowledge learned from more productive, effective communication styles, future studies are needed to focus on less active communities and comparison across all types of communities. Another potential threat in our study lies in the selection of studied platform subjects. We choose Gitter and GitHub as the representation of developers' communication and collaboration platform, respectively. However, there are other online chatting platforms like Discord and Slack, and other social coding platforms such as GitLab. Therefore, there might be discussions taking place on Discord or Slack that are not investigated in our work. However, all the selected projects are the most participated Gitter chatrooms in the corresponding domains and have achieved a good communication community on Gitter, discussing technical problems and knowledge, as reported by Shi *et al.* [2]. Other prior works also support Gitter's advantage as a communication platform for OSS [1], [27], [26]. Therefore, we believe Gitter and GitHub are able to reflect developers' open-source communication and development due to their openness, successful history management, word-wide prevalence, project-oriented organizing as well as tight association between them.

Internal Validity. The internal threats relate to experimental errors and biases. The first internal threat comes from the extraction of cross-platform contributors. Gitter provides three approaches to signing in: GitHub account,

GitLab account, and Twitter account. Signing via a GitHub account is the most dominant way (80% accounts of the selected projects are GitHub users). However, there still exists a tiny chance that developers may sign in via GitLab or Twitter accounts yet they have GitHub accounts that make contributions to the repository. The second threat might come from the process of card sorting. We understand that such a process is subject to introducing mistakes. To reduce that threat, we establish a labeling team and performed a peer review on each result. We only adopt data that received the full agreement or reach agreements on different options.

Construct Validity. The construct threats relate to the suitability of evaluation metrics. In this study, the construct threat lies in the way we construct attracting conversations and new contributors via Gitter in RQ4. We assumed developers who communicated on Gitter before their first contributions are attracted by the last discussions they had on Gitter. But some potential motivations for making the contribution might be unseen to us because we can not observe the activities out of Gitter and GitHub. However, by carefully analyzing those dialogs and determining NCs as shown in Figure 9, we consider the attracting effect is highly related.

10 Related Work

Our work is related to prior studies on analyzing developer communication platforms, analyzing OSS contributions on GitHub, and analyzing OSS activities from a multi-platform perspective.

Analyzing Developer Communication Platforms. Recently, more and more work has realized that communication platforms play an increasingly important role in OSS collaborative development. Those previous works analyzed the communication among OSS developers via different platforms, e.g., Gitter [1], [2], [8], Slack [22], Discord [10], [30], IRC [4], [23], [66], Mailing-list [67], Issue tracking systems [68], [69], and Stack Overflow [70], [71]. For example, Shi *et al.* [2] dived into developers' chat on Gitter to investigate when they interact, what community structures look like, which topics are discussed, and how they interact. Lin *et al.* [22] conducted an exploratory study on how Slack supports software engineering and team dynamics. Their research revealed that developers use Slack for personal, team-wide, and community-wide purposes, and they suggest the increasingly essential role that live chat plays. Subash *et al.* [10] proposed a dataset of Discord chat conversations from four software development communities within a year, called Disco, and analyzed five most generally discussed topics in each Discord channel. Their work emphasizes the rising popularity of Discord and builds a foundation for future studies on Discord conversations. Raglianti *et al.* [30] leveraged word cloud and source code parsing to visualize Discord interaction content to assist with software documentation and program comprehension. Shihab *et al.* [4], [23] analyzed the usage of developer IRC meeting channels of two large open-source projects from several dimensions: meeting content, meeting participants, their contribution, and meeting styles. Their results showed that IRC meetings are gaining popularity among OSS developers, and highlighted the wealth of information that can be obtained from developer chat messages. Di Sorbo *et al.* [67] proposed a taxonomy of intentions to classify sentences in developer mailing lists into six categories: feature request, opinion asking, problem discovery, solution proposal, information seeking, and information giving. Although the taxonomy has been shown to be effective in analyzing development emails and user feedback from app reviews [72], Huang *et al.* [68] found that it cannot be generalized to discussions in issue tracking systems, and they addressed the deficiencies of Di Sorbo *et al.*'s taxonomy by proposing a convolution neural network based approach. Allamanis and Sutton [70] presented a topic modeling analysis that combines question concepts, types, and code from Stack Overflow to associate programming concepts and identifiers with particular types of questions, such as, "how to perform encoding". Studies on communication platforms focus more on social interaction features among developers, such as interaction styles, community structures, raised topics, etc., without taking the development features into consideration.

Analyzing OSS Contributions on GitHub. Many prior studies analyze OSS contributions on GitHub from three aspects: motivation to contribute [73], [74], [75], [76], productivity of development [77], [78], [79], [80], and turnover of contributors [81], [82], [83], [84], [85], [86], [87]. *E.g*, two systematic literature reviews [73], [74] on motivation to join OSS have found 145 papers, and reported that proper management of motivation and satisfaction helps achieve higher productivity, avoid turnover, etc. Recently, Gerosa *et al.* [75] investigated the shifts in motivations and found that motivations related to social aspects and reputation increased in frequency. Vasilescu *et al.* [78] analyzed process data of GitHub projects, and reported that continuous integration improves the productivity of OSS project teams. Zhou *et al.* [77] explored how OSS projects on GitHub differ with regard to forking inefficiencies. They found that better modularity and centralized management are associated with more contributions, suggesting specific best practices that OSS developers can adopt to reduce forking-related inefficiencies. For developer turnover, existing literature [86], [87] reported that objective attributes of OSS projects, personal expectations, level of development experience, and conversational knowledge were associated with developer turnover. Robillard [82] highlighted that developer turnover could cause knowledge loss in OSS project teams, and Foucault *et al.* [85] reported that external turnover negatively impacts software quality. To manage and mitigate the risk of turnover-induced knowledge loss, Rigby *et al.* [84] leveraged scenario simulations and coordination requirements matrices to quantify the loss and recommend successors. Many studies have been conducted to analyze the OSS contribution based on the GitHub platform. Our work is different from the previous studies as we incorporate the Gitter platform to analyze the impact of live communication on OSS development, complementing to the existing studies.

Analyzing OSS Activities from a Multi-platform Perspective. Recently, a few studies have started investigating the impact of OSS communication on development. Sahar *et al.* [27] focused on the effect that Gitter has on GitHub issue resolution, suggesting the discussed issue reports have a longer solving time than those are never referred on Gitter, and the number of comments on GitHub issue increases after this issue was discussed on Gitter. Singh et al. [88] analyzed

the effect of knowledge sharing via Wikipedia, Forum, and Gitter on GitHub contribution, respectively. However, some of the chat rooms they used do not have a relationship with an active open source project, and they did not find evidence that the Gitter knowledge sharing can positively influence OSS development. Fang et al. [62] focused on the effect that Twitter has on GitHub project popularity and new contributors. They found that tweets have a statistically significant effect on project popularity and a small average effect on attracting new contributors. The current studies of multi-platform analysis on OSS communication and development are just emerging, and there is a lack of in-depth analysis of the relationship and impact between GitHub and Gitter. In especial, unlike other live communication platforms, Gitter is the platform that is directed toward GitHub projects. Our study bridges that gap with a large-scale analysis of the characteristics of cross-platform contributors and whether Gitter can provoke OSS development.

11 Conclusion and Future Work

In this study, we perform a cross-platform study on Gitter and GitHub in purpose to bring developers' communication and development together by investigating traits of cross-platform contributors in terms of roles and contribution, communication preference and behavioral consistency, as well as the promotive impacts of live chat on OSS in terms of new contributors, onboarded contributors and returned contributors. Our study has both theoretical and practical implications. By analyzing the empirical results, we enable a deeper understanding of the OSS developers who switch between the collaborative development platform GitHub and the instant chatting platform Gitter, and of the interplay of communication and development. We observe the positive impact of live chat on open-source development as well as attracting new contributors. Based on these findings, we provide recommendations for OSS communities and developers, as well as shed light on future research directions. In the future, we plan to utilize the multi-platform perspective to further investigate the quantitative impact of live chat, *e.g.*, to what extent can live chat affect the open-source community. Additionally, as a rising popular developers' communication platform, future attention could be paid to mine content on it along with its related GitHub community. We hope that the findings and insights that we have revealed will raise awareness of the rich information hidden in cross-platform data and enable a promising prospect for OSS communities.

12 Acknowledgements

We sincerely appreciate the anonymous reviewers for their constructive and insightful suggestions for improving this manuscript. This work was supported by the National Natural Science Foundation of China (Grant Nos.62332001, 62232016, 62072442, and 62272445), the Youth Innovation Promotion Association of the Chinese Academy of Sciences, the Basic Research Program of ISCAS (Grant No.ISCAS-JCZD-202304), the Major Program of ISCAS (Grant No.ISCAS-ZD-202302), and a grant from Huawei.

References

[1] O. Ehsan, S. Hassan, M. E. Mezouar, and Y. Zou, "An empirical study of developer discussions in the gitter platform," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 1, pp. 1–39, 2020.

[2] L. Shi, X. Chen, Y. Yang, H. Jiang, Z. Jiang, N. Niu, and Q. Wang, "A first look at developers' live chat on gitter," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 391–403.

[3] H. Hata, N. Novielli, S. Baltes, R. G. Kula, and C. Treude, "Github discussions: An exploratory study of early adoption," *Empirical Software Engineering*, vol. 27, pp. 1–32, 2022.

[4] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying the use of developer IRC meetings in open source projects," in *25th IEEE International Conference on Software Maintenance (ICSM 2009)*, 2009, pp. 147–156.

[5] E. Parra, A. Ellis, and S. Haiduc, "Gittercom: A dataset of open source developer communications in gitter," in *MSR '20: 17th International Conference on Mining Software Repositories*. ACM, 2020, pp. 563–567.

[6] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N. A. Kraft, "Exploratory study of slack q&a chats as a mining source for software engineering tools," in *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 2019, pp. 490–501.

[7] J. A. Jiang, C. Kiene, S. Middler, J. R. Brubaker, and C. Fiesler, "Moderation challenges in voice-based online communities on discord," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–23, 2019.

[8] E. Parra, M. Alahmadi, A. Ellis, and S. Haiduc, "A comparative study and analysis of developer communications on slack and gitter," *Empir. Softw. Eng.*, vol. 27, no. 2, p. 40, 2022. [Online]. Available: https://doi.org/10.1007/s10664-021-10095-1

[9] V. Stray and N. B. Moe, "Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and slack," *Journal of Systems and Software*, vol. 170, p. 110717, 2020.

[10] K. M. Subash, L. P. Kumar, S. L. Vadlamani, P. Chatterjee, and O. Baysal, "Disco: A dataset of discord chat conversations for software engineering research," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 227–231.

[11] S. Zhou, B. Vasilescu, and C. Kästner, "What the fork: A study of inefficient and efficient forking practices in social coding," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 350–361.

[12] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in github: the role of prior social links and language experience," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015, pp. 817–828.

[13] W. Xiao, H. He, W. Xu, Y. Zhang, and M. Zhou, "How early participation determines long-term sustained activity in github projects?" in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 29–41.

[14] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this github project maintained? measuring the level of maintenance activity of open-source projects," *Information and Software Technology*, vol. 122, p. 106274, 2020.

[15] Y. Yue, Y. Wang, and D. Redmiles, "Off to a good start: Dynamic contribution patterns and technical success in an oss newcomer's early career," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 529–548, 2022.

[16] J. Middleton, E. Murphy-Hill, D. Green, A. Meade, R. Mayer, D. White, and S. McDonald, "Which contributions predict whether developers are accepted into github teams," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 403–413.

[17] R. Padhye, S. Mani, and V. S. Sinha, "A study of external community contribution to open-source projects on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 332–335.

[18] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1277–1298, 2019.

[19] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proceedings of the international workshop on Principles of software evolution*, 2002, pp. 76–85.

[20] F. Calefato, M. A. Gerosa, G. Iaffaldano, F. Lanubile, and I. Steinmacher, "Will you come back to contribute? investigating the inactivity of oss core developers in github," *Empirical Software Engineering*, vol. 27, no. 3, pp. 1–41, 2022.

[21] Y. Wu, J. Kropczynski, P. C. Shih, and J. M. Carroll, "Exploring the ecosystem of software developers on github and other platforms," in *Proceedings of the Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work amp; Social Computing*, ser. CSCW Companion '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 265–268. [Online]. Available: https://doi.org/10.1145/2556420.2556483

[22] B. Lin, A. Zagalsky, M. D. Storey, and A. Serebrenik, "Why developers are slacking off: Understanding how software teams use slack," in *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2016, pp. 333–336.

[23] E. Shihab, Z. M. Jiang, and A. E. Hassan, "On the use of internet relay chat (IRC) meetings by developers of the GNOME GTK+ project," in *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE)*, 2009, pp. 107–110.

[24] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (r) evolution of social media in software engineering," in *Future of Software Engineering Proceedings*, ser. FOSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 100–116.

[25] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004, pp. 72–81.

[26] M. E. Mezouar, D. A. da Costa, D. M. German, and Y. Zou, "Exploring the use of chatrooms by developers: An empirical study on slack and gitter," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 3988–4001, 2022.

[27] H. Sahar, A. Hindle, and C.-P. Bezemer, "How are issue reports discussed in gitter chat rooms?" *Journal of Systems and Software*, vol. 172, p. 110852, 2021.

[28] M. Raglianti, C. Nagy, R. Minelli, B. Lin, and M. Lanza, "On the rise of modern software documentation (pearl/brave new idea)," in *37th European Conference on Object-Oriented Programming (ECOOP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[29] V. Ebert, D. Graziotin, and S. Wagner, "How are communication channels on github presented to their intended audience?–a thematic analysis," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, 2022, pp. 40–49.

[30] M. Raglianti, C. Nagy, R. Minelli, and M. Lanza, "Using discord conversations as program comprehension aid," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, 2022, pp. 597–601.

[31] J. K. Kummerfeld, S. R. Gouravajhala, J. Peper, V. Athreya, C. Gunasekara, J. Ganhotra, S. S. Patel, L. Polymenakos, and W. S. Lasecki, "A large-scale corpus for conversation disentanglement," *arXiv preprint arXiv:1810.11118*, 2018.

[32] Gitter, "Rest api," https://developer.gitter.im/docs/rest-api, 2020.

[33] GitHub, "Rest api," https://docs.github.com/en/rest, 2022.

[34] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in github issue and pr comments," *Journal of Systems and Software*, vol. 175, p. 110911, 2021.

[35] M. Wessel, B. M. de Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, "The power of bots: Characterizing and understanding bots in oss projects," vol. 2, no. CSCW, nov 2018.

[36] B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of GitHub teams," in *12th Working Conference on Mining Software Repositories, Data Track*, ser. MSR. IEEE, 2015, pp. 514–517.

[37] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ser. MSR '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 137–143.

[38] GitHub, "Github docs," https://docs.github.com/, 2022.

[39] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: An empirical study on count and network metrics," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 164–174.

[40] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[41] T. T. Dinh-Trong and J. M. Bieman, "The freebsd project: A replication case study of open source development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, 2005.

[42] J. Coelho, M. T. Valente, L. L. Silva, and A. Hora, "Why we engage in floss: Answers from core developers," in *proceedings of the 11th international workshop on cooperative and human aspects of software engineering*, 2018, pp. 114–121.

[43] K. Crowston, K. Wei, Q. Li, and J. Howison, "Core and periphery in free/libre and open source software team communications," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, vol. 6. IEEE, 2006, pp. 118a–118a.

[44] J. Cheng and J. L. Guo, "Activity-based analysis of open source software contributors: Roles and dynamics," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2019, pp. 11–18.

[45] M. Palyart, G. C. Murphy, and V. Masrani, "A study of social interactions in open source component use," *IEEE Transactions on Software Engineering*, vol. 44, no. 12, pp. 1132–1145, 2017.

[46] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[47] A. E. Mislove, *Online social networks: measurement, analysis, and applications to distributed information systems*. Rice University, 2009.

[48] X. Zhao, F. Liu, J. Wang, and T. Li, "Evaluating influential nodes in social networks by local centrality with a coefficient," *ISPRS International Journal of Geo-Information*, vol. 6, no. 2, p. 35, 2017.

[49] I. Himelboim, "Reply Distribution in Online Discussions: A Comparative Network Analysis of Political and Health Newsgroups," *Journal of Computer-Mediated Communication*, vol. 14, no. 1, pp. 156–177, 10 2008.

[50] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 3, no. 1, 2009.

[51] D. Chen, L. Lü, M.-S. Shang, Y.-C. Zhang, and T. Zhou, "Identifying influential nodes in complex networks," *Physica a: Statistical mechanics and its applications*, vol. 391, no. 4, pp. 1777–1787, 2012.

[52] S. Beyer, C. Macho, M. Pinzger, and M. D. Penta, "Automatically classifying posts into question categories on stack overflow," in *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018*. ACM, 2018, pp. 211–221.

[53] M. Antikainen, T. Aaltonen, and J. Väisänen, "The role of trust in oss communities — case linux kernel community," in *IFIP International Conference on Open Source Systems*. Springer, 2007, pp. 223–228.

[54] A. Mockus, "Insights from open source software supply chains (keynote)," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds. ACM, 2019, p. 3. [Online]. Available: https://doi.org/10.1145/3338906.3342813

[55] C. Spearman, ""general intelligence" objectively determined and measured." 1961.

[56] G. Rugg and P. McGeorge, "The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts," *Expert Systems*, vol. 14, no. 2, pp. 80–93, 1997.

[57] J. Marsan, M. Templier, P. Marois, B. Adams, K. Carillo, and G. L. Mopenza, "Toward solving social and technical problems in open source software ecosystems: using cause-and-effect analysis to disentangle the causes of complex problems," *IEEE Software*, vol. 36, no. 1, pp. 34–41, 2018.

[58] A. Mockus, "Organizational volatility and its effects on software defects," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 117–126.

[59] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015, pp. 829–841.

[60] J. W. Tukey *et al.*, *Exploratory data analysis*. Reading, MA, 1977, vol. 2.

[61] A. A. Markov, "Extension of the limit theorems of probability theory to a sum of variables connected in a chain," *Dynamic probabilistic systems*, vol. 1, pp. 552–577, 1971.

[62] H. Fang, H. Lamba, J. D. Herbsleb, and B. Vasilescu, ""this is damn slick!" estimating the impact of tweets on open source project popularity and new contributors," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 2116–2129. [Online]. Available: https://doi.org/10.1145/3510003.3510121

[63] C. Wild and G. Seber, "The wilcoxon rank-sum test," *Chance Encounters: A First Course in Data Analysis and Inference*, vol. 611, 2011.

[64] F. Fagerholm, A. Sanchez Guinea, J. Borenstein, and J. Münch, "Onboarding in open source projects," *IEEE Software*, vol. 31, no. 6, pp. 54–61, 2014.

[65] H. Fang, B. Vasilescu, H. Lamba, and J. Herbsleb, ""this is damn slick!" estimating the impact of tweets on open source project popularity and new contributors," 2022.

[66] L. Yu, S. Ramaswamy, A. Mishra, and D. Mishra, "Communications in global software development: An empirical study using GTK+ OSS repository," in *Proceedings of the 2011th Confederated International Conference on the Move to Meaningful Interest Systems, OTM'11*, 2011, pp. 218–227.

[67] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall, "Development emails content analyzer: Intention mining in developer discussions (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, 2015, pp. 12–23.

[68] Q. Huang, X. Xia, D. Lo, and G. C. Murphy, "Automating intention mining," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2018.

[69] D. Arya, W. Wang, J. L. C. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019*, 2019, pp. 454–464.

[70] M. Allamanis and C. Sutton, "Why, when, and what: Analyzing stack overflow questions by topic, type, and code," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13*. IEEE Computer Society, 2013, pp. 53–56.

[71] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using stack overflow," *Empir. Softw. Eng.*, vol. 21, no. 3, pp. 1192–1223, 2016.

[72] S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," pp. 281–290, 2015.

[73] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Inf. Softw. Technol.*, vol. 50, no. 9-10, pp. 860–878, 2008. [Online]. Available: https://doi.org/10.1016/j.infsof.2007.09.004

[74] A. C. C. França, T. B. Gouveia, P. C. F. Santos, C. A. Santana, and F. Q. B. da Silva, "Motivation in software engineering: A systematic review update," in *15th International Conference on Evaluation & Assessment in Software Engineering, EASE 2011, Durham, UK, 11-12 April 2011, Proceedings*. IET - The Institute of Engineering and Technology / IEEE Xplore, 2011, pp. 154–163. [Online]. Available: https://doi.org/10.1049/ic.2011.0019

[75] M. A. Gerosa, I. Wiese, B. Trinkenreich, G. Link, G. Robles, C. Treude, I. Steinmacher, and A. Sarma, "The shifting sands of motivation: Revisiting what drives contributors in open source," *CoRR*, vol. abs/2101.10291, 2021. [Online]. Available: https://arxiv.org/abs/2101.10291

[76] F. Q. B. da Silva and A. C. C. França, "Towards understanding the underlying structure of motivational factors for software engineers to guide the definition of motivational programs," *J. Syst. Softw.*, vol. 85, no. 2, pp. 216–226, 2012. [Online]. Available: https://doi.org/10.1016/j.jss.2010.12.017

[77] S. Zhou, B. Vasilescu, and C. Kästner, "What the fork: a study of inefficient and efficient forking practices in social coding," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds. ACM, 2019, pp. 350–361. [Online]. Available: https://doi.org/10.1145/3338906.3338918

[78] B. Vasilescu, Y. Yu, H. Wang, P. T. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, E. D. Nitto, M. Harman, and P. Heymans, Eds. ACM, 2015, pp. 805–816. [Online]. Available: https://doi.org/10.1145/2786805.2786850

[79] S. Khatoonabadi, D. E. Costa, R. Abdalkareem, and E. Shihab, "On wasted contributions: Understanding the dynamics of contributor-abandoned pull requests," *CoRR*, vol. abs/2110.15447, 2021. [Online]. Available: https://arxiv.org/abs/2110.15447

[80] A. Ram, A. A. Sawant, M. Castelluccio, and A. Bacchelli, "What makes a code change easier to review: an empirical investigation on code change reviewability," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, G. T. Leavens, A. Garcia, and C. S. Pasareanu, Eds. ACM, 2018, pp. 201–212. [Online]. Available: https://doi.org/10.1145/3236024.3236080

[81] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Trans. Software Eng.*, vol. 47, no. 6, pp. 1277–1298, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2918536

[82] M. P. Robillard, "Turnover-induced knowledge loss in practice," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 1292–1302. [Online]. Available: https://doi.org/10.1145/3468264.3473923

[83] Y. Zhang, H. Liu, X. Tan, M. Zhou, Z. Jin, and J. Zhu, "Turnover of companies in openstack: Prevalence and rationale," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, pp. 75:1–75:24, 2022. [Online]. Available: https://doi.org/10.1145/3510849

[84] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss: Case studies of chrome and a project at avaya," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, L. K. Dillon, W. Visser, and L. A. Williams, Eds. ACM, 2016, pp. 1006–1016. [Online]. Available: https://doi.org/10.1145/2884781.2884851

[85] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J. Falleri, "Impact of developer turnover on quality in open-source software," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, E. D. Nitto, M. Harman, and P. Heymans, Eds. ACM, 2015, pp. 829–841. [Online]. Available: https://doi.org/10.1145/2786805.2786870

[86] Y. Yu, A. Benlian, and T. Hess, "An empirical study of volunteer members' perceived turnover in open source software projects," in *45th Hawaii International International Conference on Systems Science (HICSS-45 2012), Proceedings, 4-7 January 2012, Grand Wailea, Maui, HI, USA*. IEEE Computer Society, 2012, pp. 3396–3405. [Online]. Available: https://doi.org/10.1109/HICSS.2012.97

[87] A. Schilling, S. Laumer, and T. Weitzel, "Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in FLOSS projects," in *45th Hawaii International International Conference on Systems Science (HICSS-45 2012), Proceedings, 4-7 January 2012, Grand Wailea, Maui, HI, USA*. IEEE Computer Society, 2012, pp. 3446–3455. [Online]. Available: https://doi.org/10.1109/HICSS.2012.644

[88] V. K. Singh, S. Chakraborty, and A. Kadian, "The effect of knowledge sharing on open source contribution: A multi-platform perspective," in *53rd Hawaii International Conference on System Sciences, HICSS 2020, Maui, Hawaii, USA, January 7-10, 2020*. ScholarSpace, 2020, pp. 1–10. [Online]. Available: https://hdl.handle.net/10125/64088