A First Look at Developers' Live Chat on Gitter

Lin Shi
shilin@iscas.ac.cn
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences, China

Xiao Chen
chenxiao2021@iscas.ac.cn
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences, China

Ye Yang
yyang4@stevens.edu
School of Systems and Enterprises, Stevens Institute of Technology
Hoboken, NJ, USA

Hanzhi Jiang
Ziyou Jiang
{hanzhi2021,ziyou2019}@iscas.ac.cn
Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences, China

Nan Niu
nan.niu@uc.edu
Department of EECS, University of Cincinnati, Cincinnati, OH
USA

Qing Wang*
wq@iscas.ac.cn
State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences, China

ABSTRACT

Modern communication platforms such as Gitter and Slack play an increasingly critical role in supporting software teamwork, especially in open source development. Conversations on such platforms often contain intensive, valuable information that may be used for better understanding OSS developer communication and collaboration. However, little work has been done in this regard. To bridge the gap, this paper reports a first comprehensive empirical study on developers' live chat, investigating when they interact, what community structures look like, which topics are discussed, and how they interact. We manually analyze 749 dialogs in the first phase, followed by an automated analysis of over 173K dialogs in the second phase. We find that developers tend to converse more often on weekdays, especially on Wednesdays and Thursdays (UTC), that there are three common community structures observed, that developers tend to discuss topics such as API usages and errors, and that six dialog interaction patterns are identified in the live chat communities. Based on the findings, we provide recommendations for individual developers and OSS communities, highlight desired features for platform vendors, and shed light on future research directions. We believe that the findings and insights will enable a better understanding of developers' live chat, pave the way for other researchers, as well as a better utilization and mining of knowledge embedded in the massive chat history.

CCS CONCEPTS

• **Software and its engineering → Open source model**; • **General and reference → Empirical studies**.

*Corresponding author.

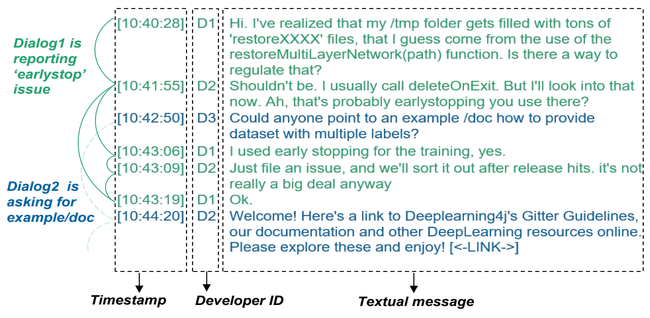
KEYWORDS

Live chat, Team communication, Open source, Empirical Study

ACM Reference Format:
Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyou Jiang, Nan Niu, and Qing Wang. 2021. A First Look at Developers' Live Chat on Gitter. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3468264.3468562

1 INTRODUCTION

More than ever, online communication platforms, such as Gitter, Slack, Microsoft Teams, Google Hangout, and Freenode, play a fundamental role in team 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 asynchronous communication like emails in some cases [40, 58, 59]. This is especially true for open source projects that are contributed by globally distributed developers, as well as for many companies allowing developers to work from home due to the COVID-19 pandemic. Conversations from online communication platforms contain rich information for studying developer behaviors. Figure 1 exemplifies a slice of live chat log from the *Deeplearning4j* Gitter community. Each utterance consists of a timestamp, developer ID, and a textual message. In addition, two dialogs are embedded in the chat log. The first one is reporting an issue about 'earlystop', and the second one is asking for documentation support. Thus, valuable information, such


Dialog1 is reporting 'earlystop' issue
Dialog2 is asking for example/doc
Timestamp
Developer ID
Textual message

```
[10:40:28] D1 Hi. I've realized that my /tmp folder gets filled with tons of
              'restoreXXXX' files, that I guess come from the use of the
              restoreMultiLayerNetwork(path) function. Is there a way to
              regulate that?
[10:41:55] D2 Shouldn't be. I usually call deleteOnExit. But I'll look into that
              now. Ah, that's probably earlystopping you use there?
[10:42:50] D3 Could anyone point to an example /doc how to provide
              dataset with multiple labels?
[10:43:06] D1 I used early stopping for the training, yes.
[10:43:09] D2 Just file an issue, and we'll sort it out after release hits. it's not
              really a big deal anyway
[10:43:19] D1 Ok.
[10:44:20] D2 Welcome! Here's a link to Deeplearning4j's Gitter Guidelines,
              our documentation and other DeepLearning resources online.
              Please explore these and enjoy! [<-LINK->]
```

Figure 1: A slice of live chat log from the DL4J community.

as when OSS developers interact, what are community structures, which topics are discussed, and how OSS developers interact, can be derived from the massive live chat data, which are important for learning knowledge from productive and effective communication styles, improving existing live chat platforms, and guiding research directions on promoting efficient and effective OSS collaboration.

Despite that a few empirical studies started to advocate the usefulness of the conversations for understanding developer behaviors [40, 59, 67], little focuses on how and what the developers communicate in live chat. The most related research is reported by Shihab *et al.* [58] on Internet Relay Chat to analyze content, participants, and styles of communications. However, their subjects are IRC meeting logs, which are different in many aspects from developer live chat conversations. Another thread of related work by Parra *et al.* [49] and Chatterjee *et al.* [15] presents two datasets of open source developer communications in Gitter and Slack respectively, with the purpose of highlighting that live developer communications are untapped information resources. This motivates our study to derive a deeper understanding about the nature of developer communications in open-source software.

In this paper, we conduct a first comprehensive empirical study on developers' live chat on Gitter, investigating four characteristics: when they interact (communication profile), what community structures look like (community structure), which topics are discussed (discussion topic), and how they interact (interaction pattern). To that end, we first collect a large scale of developer daily chat from eight popular communities. Then we manually disentangle 749 dialogs, and select the best disentanglement model from four state-of-the-art models according to their evaluation results on the 749 dialogs. After automatic disentanglement, we perform an empirical study on live chat aiming to reveal four characteristics: communication profile, community structure, dialog topic, and interaction pattern. In total, we studied 173,278 dialogs, 1,402,894 utterances, contributed by 95,416 users from eight open source communities. The main results include: (1) Developers are more likely to chat on workdays than weekends, especially on Wednesday and Thursday (UTC); (2) Three social patterns are observed in the OSS community of live chat: Polaris network, Constellation network, and Galaxy network; (3) The top three topics that developers frequently discuss in live chat are API usages, errors, and background information; and (4) Six interaction patterns are identified in live chat including exploring solution, clarifying answer, clarifying question, direct/discussed answer, self-answered monologue, and unanswered monologue. The major contributions of this paper are as follows.

- We conduct a first large scale analysis study on developers' live chat messages, providing empirically-based quantitative and qualitative results towards better understanding developer communication profiles, community structures, discussion topics, and interaction patterns.
- We provide practical insights on productive dialogs for individual developers and OSS communities, highlight desired features for platform vendors, and shed light on future directions for researchers.
- We provide a large-scale dataset[1] of live chat to facilitate the replication of our study and future applications.

[1]https://github.com/LiveChat2021/LiveChat#5-download

In the remainder of the paper, Section II illustrates the background. Section III presents the study design. Section IV describes the results and analysis. Section V is the discussion of results and threats to validity. Section VI introduces the related work. Section VII concludes our work.

2 BACKGROUND

This section describes related key concepts and technologies.

2.1 The Gitter Platform

Many OSS communities utilize Gitter [31] or Slack [32] as their live communication means. In particular, Gitter is currently the most popular online communication platform [37] since it provides open access to public chat rooms and free access to historical data [49]. Considering the popular, open, and free access nature, we conduct this study based on Gitter[2]. Communities in Gitter usually have multiple chatting rooms: one general room and several specific-topic rooms. Typically, the general room contains most of the participants. In this study, we only focus on the general rooms. In total, there are 2,171 communities in Gitter that can be publicly accessed. The total number of participants of the 2,171 communities is 733,535 as of Nov. 20, 2020.

Three main concepts about Gitter live chat log are concerned in the scope of this study, including *chat log, utterance, and dialog*. In Gitter, developer conversations in one chatting room are recorded in a chat log. As illustrated in Figure 1, a typical live chat log contains a sequential set of utterances in chronological order. Each *utterance* consists of a timestamp, developer id, and a textual message initiating a question or responding to an earlier message. A *chat log* typically contains a large number of utterances, and at any given time, multiple consecutive utterances might be possibly responding to different threads of dialog discussions. The interleaving nature of utterances leads to entangled *dialogs*, as illustrated with the two colors in Figure 1. The two colors are used to highlight utterances belonging to two different dialogs, where the links between two utterances indicate the responding relationship.

2.2 Challenges in Chat Analysis

Different from many other sources of software development related communication, the information on online communication platforms are shared in an unstructured, informal, and interleaved manner. Thus, analyzing live chat is quite challenging due to the following barriers. **(1) Entangled dialogs**. Utterances in chat logs form stream information, with dialogs often entangling such as a single conversation is interleaved with other dialogs, as shown in Figure 1. It is difficult to perform any kind of high-level dialog analysis without dividing utterances into a set of distinct dialogs. **(2) Expensive human effort.** Chat logs are typically high-volume and contain informal dialogs covering a wide range of technical and complex topics. Analyzing these dialogs requires experienced analysts to spend a large amount of time so that they can understand the dialogs thoroughly. Thus, it is very expensive to conduct a comprehensive study on developers' live chat. **(3) Noisy data**. There exist noisy utterances such as duplicate and unreadable messages in

[2]In Slack, communities are controlled by the team administrators, whereas in Gitter, access to the chat data is public.

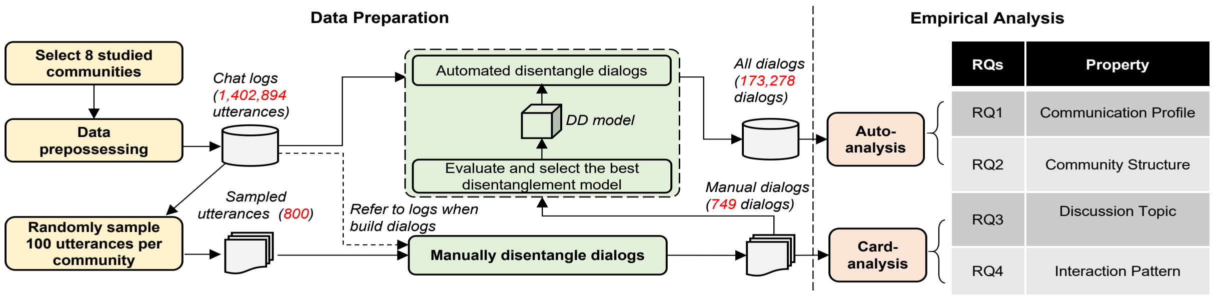

Data Preparation
Empirical Analysis
Select 8 studied communities
Data prepossessing
Randomly sample 100 utterances per community
Chat logs (1,402,894 utterances)
Sampled utterances (800)
Automated disentangle dialogs
DD model
Evaluate and select the best disentanglement model
Refer to logs when build dialogs
Manually disentangle dialogs
All dialogs (173,278 dialogs)
Manual dialogs (749 dialogs)
Auto-analysis
Card-analysis
RQs
Property
RQ1
Communication Profile
RQ2
Community Structure
RQ3
Discussion Topic
RQ4
Interaction Pattern

Figure 2: Overview of research methodology

chat logs that do not provide any valuable information. The noisy data poses a difficulty to analyze and interpret the communicative dialogs. Next, we will introduce several existing techniques that can automatically disentanglement dialogs.

2.3 Dialog Disentanglement (DD)

Four state-of-the-art techniques have been proposed to address the entangled dialog challenge in the natural language processing area: (1) **BiLSTM model** [33] predicts whether there exists an edge between two utterances, where the edge means one utterance is a response to another. It employs a bidirectional recurrent neural network with 160 context maximum size, 200 neurons with one hidden layer. The input is a sequence of 512-dimensional word vectors; (2) **BERT model** [19] predicts the probability of utterance u_i's clustering label under the context utterances $u_{\leq i}$ and labels $y_{<i}$. It utilizes the *Masked Language Model* and *Next Sentence Prediction* [19] to encode the input utterances, with 512 embedding size and 256 hidden units; (3) **E2E model** [41] performs the dialog Session-State encoder to predict dialog clusters, with 512 embedding size, 256 hidden neurons and 0.05 noise ratio; and (4) **FF model** [39] is a feed-forward neural network with two layers, 256-dimensional hidden vectors, and softsign non-linearities. The input is a 77-dimensional numerical feature extracted from the utterance texts, which includes TF-IDF, user name, time interval, whether two utterances contain the same words, *etc.* In addition, there are four clustering metrics that are widely used for DD evaluation: Normalized Mutual Information (NMI) [61], Adjusted Rand Index (ARI) [54], Shen-F value [56] and F1 score [18].

3 METHODOLOGY AND STUDY DESIGN

This study aims to investigate four research questions:

RQ1 (Communication Profile): *Do Gitter communities demonstrate consistent community communication profiles?* This research question aims at examining common communication profiles across the eight communities, particularly the frequent time-frames that the developers are active and the typical time interval of a dialog.

RQ2 (Community Structure): *What are the structural characteristics of social networks built from developer live chat data?* To understand community characteristics of live chat networks, we perform social network analysis on live-chat utterances, in which each developer is treated as a node, with edges defined as a pair of developers co-occurring in one or more dialogs.

RQ3 (Dialog Topic): *What are the primary topic types frequently discussed by developers in live chat?* This research question is designed to identify discussion topics in developers' live chat. There have been studies analyzing discussion topics in open forums [4], emails [34], and posts in Stack Overflow [9, 35]. It remains unknown what developers are talking about in live chat. This study aims at filling the gap and providing a complementary perspective using live-chat as a new data source.

RQ4 (Interaction Pattern): *How do developers typically interact with each other in live chat?* This research question intends to uncover underlying interaction patterns which signify how developers typically interact (*e.g.*, initiate discussion, respond to questions and social chat.) with one another throughout a dialog life cycle.

3.1 Methodology Overview

The research methodology follows two phases, as illustrated in Figure 2. First, in the data preparation phase, a large scale of developer daily chat utterances data are collected from eight active communities, and the raw chat utterances data are processed and transformed into associated dialogs using two approaches, *i.e.*, manual screening on a randomly sampled small dataset and automated analysis employing the identified best DD model on the whole dataset. Second, in the empirical analysis phase, further analysis will be designed and conducted on these two datasets in order to investigate the characteristics and develop a better understanding of developer live chat data with respect to the research questions.

3.2 Data Preparation

Studied communities. To identify studied communities, we select the Top-1 most participated communities from eight active domains, covering front-end framework, mobile, data science, DevOps, blockchain platform, collaboration, web app, and programming language. Then, we collect the daily chat utterances from

Table 1: The statistics of the studied Gitter communities

Community	Domain	Entire Population			Sample Population		
		P	D	U	P	D	U
Angular[30]	Frontend Framework	22,467	79,619	695,183	125	97	778
Appium[25]	Mobile	3,979	4,906	29,039	73	87	724
DL4J[27]	Data Science	8,310	27,256	252,846	93	100	1,130
Docker[21]	DevOps	8,810	3,954	22,367	74	90	1,126
Ethereum[24]	Blockchain Platform	16,154	17,298	91,028	116	96	516
Gitter[42]	Collabration Platform	9,260	7,452	34,147	87	86	515
Nodejs[16]	Web Application Framework	18,118	13,981	81,771	144	98	737
Typescript[17]	Programming Language	8,318	18,812	196,513	110	95	1,700
Total		*95,416*	*173,278*	*1,402,894*	*822*	*749*	*7,226*

these communities. Gitter provides REST API [28] to get data about chatting rooms and post utterances. In this study, we use the REST API to acquire the chat utterances of the eight selected communities, and the retrieved dataset contains all utterances as of "2020-11-20". Detailed statistics are shown in Table 1, where P refers to the number of participants, D refers to the number of dialogs, and U refers to the number of utterances. The total number of participants for the eight communities is 95,416, accounting for 13% of the total population in Gitter. Thus, we consider that the eight communities are representative of the Gitter platform.

Prepossessing the textual utterances. We first normalize non-ASCII characters like emojis to standard ASCII strings. Some low-frequency tokens contribute little to the analysis of live chat, such as URL, email address, code, HTML tags, and version numbers. We replace them with specific tokens *<URL>, <EMAIL>, <HTML>, <CODE>, and <ID>*. We utilize Spacy [2] to tokenize sentences into terms and perform lemmatization and lowercasing on terms with Spacy to alleviate the influence of word morphology.

Manual labeling of dialog disentanglement. We employ a 3-step manual process to generate a sample dialog dataset for further analysis. First, we randomly sample 100 utterances from each community's live chat log, with the intention to trace corresponding dialogs associated with each of the 100 utterances. This step leads to a total of 800 utterances from the entire 1,402,894 utterances of the eight communities. Next, using each utterance as a seed, we identify its preceding and successive utterances iteratively so that we can group related utterances into the same dialog as complete as possible. Specifically, for each utterance, we determine its context by examining the consecutive chats in the chat log, and manually link it to related utterances belonging to the same dialog. Then, the next step is cleaning. Specifically, we excluded unreadable dialogs: (1) dialogs that are written in non-English languages; (2) dialogs that contain too much code or stack traces; (3) Low-quality dialogs such as dialogs with many typos and grammatical errors; and (4) Dialogs that involve channel robots. After these steps, we include additional six thousand utterances which are associated with the initial 800 utterances. This leads to a total of 7,226 utterances, manually disentangled into 749 dialogs, as summarized in Table 1. Note that, removing bot-involved dialogs has little impact on our results. First, the bot-involved dialogs are relatively small in volume. In our study, only one of the eight projects utilizes bots, and more specifically, only nine out of 800 sampled dialogs are excluded due to bot involvement. Second, we observed that the bot-generated utterances are rather trivial, such as greeting information, links to general guidelines, and status updates.

To ensure the correctness of the disentanglement results, a labeling team was put together, consisting of one senior researcher and six 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 senior researcher trained the six Ph.D. candidates on how to disentangle dialogs and provided consultation during the process. The disentanglement results from the Ph.D. candidates were reviewed by others. We only accepted and included dialogs to our dataset when the dialogs received full agreement. When a dialog received different disentanglement results, we hosted a discussion with all team members to decide through voting. The average Cohen's Kappa about dialog disentanglement is 0.81.

Automated Dialog Disentanglement. To analyze the dialogs on a large scale, we experiment with the four state-of-art DD approaches, as introduced in Section 2.2 (*i.e.* BiLSTM model, Bert model, E2E model, and FF model). Specifically, we use the manual disentanglement sample data from the previous step as ground truth data, compare and select the best DD model for the purpose of further analysis in this study. The comparison results from our experiments show that the **FF approach** significantly outperforms the others on disentangling developer live chat by achieving the highest scores on all the metrics. The average scores of NMI, Shen-F, F1, and ARI are 0.74, 0.81, 0.47, and 0.57 respectively[3]. Finally, we use the best FF model to disentangle all the 1,402,894 utterances in chat logs. In total, we obtain 173,278 dialogs.

3.3 Empirical Analysis Design

3.3.1 Analysis for RQ1 (Communication Profile). As good communication habits suggest more productive development practices, we intend to reveal the temporal communication profiles of developers, including when the developers are active and how long the respondent replies to the dialog initiator. First, we collect all the utterance time of the entire population, and analyze the peak hours and peak days. Then we calculate the response time lag :

$$Time\_lag = T_r - T_i \tag{1}$$

where T_i is the time that the initiator launched the dialog, and T_r is the time the first respondent replied. We automatically calculate the utterance times and response time lags for all the 173,278 dialogs.

3.3.2 Analysis for RQ2 (Community Structure). We aim to visualize the social networks of developers in live chat, and summarize the common structures. Social network analysis (SNA) describes relationships among social entities, as well as the structures and implications of their connections [65]. For studying relationships among developers in one OSS community, we generate the social networks according to the following definition:

$$\begin{aligned} G &= \{V, E\} \\ V &= \{d_1, d_2, ..., d_n\} \\ E &= \{< d_j, d_k >\} \end{aligned} \tag{2}$$

where d_i is a developer in the chatting room, d_j is one dialog initiator, and d_k is a respondent to d_j. Specifically, for each disentangled dialog, we first identify its initiator and all the respondents. The initiator is the developer who launches the dialog, and the respondents are other developers who participate in the dialog. Then we add a link between the initiator and each respondent. (Note that, RQ2 focuses on exploring responding behaviors, i.e., the interaction between initiators and respondents, thus the links/edges are defined only between these two roles. Additionally, we will explore the interaction relationship among initiators and all responders in RQ4, which focuses on discussion behaviors.) Finally, we employ an unweighted graph when constructing the social networks, for visualizing the relationship of all the developers in live chat. The social network could exhibit the connectivity and density of the

[3]Due to space, experimental details on evaluation existing DD models are provided on Github: https://tinyurl.com/3dyu5n44

open-source community. We build the social networks based on all the 173,278 dialogs by using the automatic graph tool, Gephi [6].

To understand the topology of the eight social networks, we report the following SNA measures that have been widely used by previous studies [45, 55]. Note that, we excluded developers who never received replies (AKA. Haircut) when calculating SNA measures following previous work [11, 62]. (1) **Degree** [20] measures the number of edges. (2) **Betweenness** [26] measures the frequency that a developer lies on the shortest path between other developers. (3) **Closeness** [7] measures the average farness (inverse distance) of one developer to all other developers. (4) **Diameter** [13] is the largest geodesic distance in the connected network. (5) **Clustering coefficient** [66] is a measure of the degree to which developers in a graph tend to cluster together.

3.3.3 Analysis for RQ3 (Discussion Topic). Our goal is identifying discussion topics in developers' live chat. Chatterjee *et al.* [15] observed that live chats provide similar information as can be found in Q&A posts on Stack Overflow. To effectively identify and organize dialog topics, we extend Beyer *et.al*'s category of question categories on Stack Overflow [9]. We choose Beyer *et.al*'s category for two reasons. First, since developers use both Question and Answer forums (such as Stack Overflow) and live chat to resolve development issues, we consider the categories of Stack Overflow questions are partially applicable to the dialog topics in live chat. Second, Beyer *et.al*'s category harmonizes five taxonomies presented in previous studies [3, 8, 10, 51, 64] that are already validated and suitable to the posts of developers' questions.

Nonetheless, the predefined category is not meant to be comprehensive, thus, we employ a *hybrid card sort* process [22] to manually determine the topics of dialogs. In a *hybrid card sort*, the sorting begins with the predefined Beyer *et.al*'s category and participants could create their own as well. The newly-created topic is instantly updated into the topic set and can be used by other participants then. The participants are the same team that manually disentangle dialogs, and the labeling process is similar to manual dialog disentanglement as introduced in Section 3.2. Specifically, the sorting process is conducted in one round, with a concluding discussion session to resolve the disagreement in labels based on majority voting. The average Cohen's Kappa about dialog topics is 0.86.

3.3.4 Analysis for RQ4 (Interactive Pattern). Live-chat conversations generally serve the purposes of solution exploration and discussion stimulation. To uncover underlying patterns that shape and/or direct more productive conversations, we first adopt a developer intent codebook [50] and manually label the interaction links that appeared in each dialog. The developer intent codebook is built from previous work on user intent in information-seeking conversations [50], as summarized in Table 2.

Table 2: Developer intent category in live chat

Code	Label	Description
OQ	Original Question	The first question from the developer to initiate the dialog
CQ	Clarifying Question	Developers ask for clarifications
FD	Further Details	Developers provide more details
FQ	Follow Up Question	Developers ask for follow up questions about relevant issues
PA	Potential Answer	A potential answer or solution provided by developers
PF	Positive Feedback	Developer provides positive feedback for working solutions
NF	Negative Feedback	Developer provides negative feedback for useless solutions
GG	Greetings/Gratitude	Greetings or expressing gratitude

Then, we employ an *open card sort* [52] process to assign an interactive pattern to a dialog based on the sequence of developers' intents. In an open sort, the sorting begins with no predefined patterns and participants develop their own patterns. The two participants individually assigned patterns to the same dialogs. The sorting process is conducted in multiple rounds. In the first round, all participants label dialogs of one community, with an intensive discussion session to achieve conceptual coherence about patterns. A shared pool of patterns is utilized and carefully maintained, and each participant could select existing patterns from and/or add new pattern names into the shared pool. Then we divide into two teams to label the remaining dialogs. Each dialog will receive two pattern labels, and we resolve disagreement based on majority voting. The average Cohen's Kappa about interactive patterns is 0.82.

After identifying underlying interaction patterns, we further explore their statistical characteristics in aspects of distribution and duration. We calculate the duration of a dialog as follows:

$$Duration = T_e - T_i \tag{3}$$

where T_e is the time that the dialog ended, and T_i is the time that the initiator launched the dialog. This metric can reflect the life cycle of one dialog.

Note that, to keep the workload of manually labeling each dialog manageable, we answer RQ3 and RQ4 by manually analyzing the 749 sampled dialogs. We believe that although we could only manually analyze a small percentage of the disentangled dialogs, this dataset supports our methodology as being useful for discovering valuable findings.

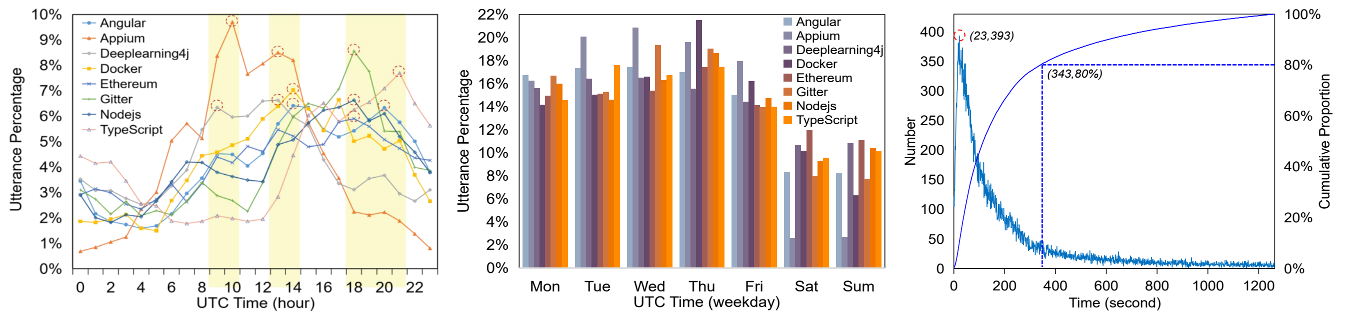
4 RESULTS AND ANALYSIS

4.1 RQ1: Communication Profile

To answer this question, we analyze two metrics, *i.e.* utterance time and response time. Next, we report the results of comparing these metrics across the eight Gitter communities.

Utterance Time. Figure 3(a) compares the distribution of utterances' intensity over 24 hours, across the eight communities. First, we identify the peak hours of each community in red dashed circles, then highlight the time windows based on the peak hours contained in it with the yellow shade. We can see that, there are three windows of peak hours, which are from UTC 9 to 10, 13 to 14, and 18 to 21. In addition, UTC 1 to 6 corresponds to the low chatting-activity hours. Developers are less active in chatting at that time. Figure 3(b) shows the distribution of the utterances across different weekdays. We can see that developers chat more on workdays than on weekends (UTC).

Response Time. Figure 3(c) exhibits the distribution of response time calculated from the 173,278 dialogs of the eight communities. The average response time is 220 seconds, the maximum time lag is 1,264 seconds, and the minimum time lag is two seconds. The peak point is (23, 393), which means there are 393 dialogs that got replies in 23 seconds. We can see that, the time lag largely increases from 0 to 23 seconds, and descend in a long tail. Eighty percent of the dialogs get first responses in 343 seconds. As reported by a recent study on Stack Overflow [43], the threshold of fast answers was 439 seconds. In comparison, live chat gets 50% faster ((439-220)/439)



(a) Hourly distribution (b) Day of week distribution (c) Frequency distribution of response times

Figure 3: Statistic results about communication profiles

replies than the fast answers in Stack Overflow. Therefore, we consider the responses from the live chat are relatively fast.

Answering RQ1: The peak hours for live chat are from UTC 9 to 10, 13 to 14, and 18 to 21, while UTC 1 to 6 is the low-active hours. Developers are more likely to chat on workdays than weekends, especially on Wednesdays and Thursdays (UTC). Moreover, live chat gets 50% faster replies than the fast answers in Stack Overflow.

4.2 RQ2: Community Structure

To answer RQ2, we first examine the structural properties of the developer social networks across the eight communities, and then try to draw some common observations based on these social networks.

Properties of social networks. Table 3 shows the social network properties of the eight communities. Init.%, Resp.%, and Both% denote the percentage of developers serving the role of dialog initiators, respondents, and both. Intuitively, we consider that respondents share their knowledge with others, while initiators receive knowledge from others. We can see that, the four communities (Appium, Docker, Gitter, and Ethereum) have a higher percentage (75.04%-81.70%) of dialog initiators and a lower percentage (18.30%-24.96%) of respondents/both. The high percentage of dialog initiators may relate to the applicable nature of the open-source projects, *e.g.*, Ethereum is one of the most widely used open-source blockchain systems, thus there are a large number of users acquiring technical support from live chat. The other four communities (Angular, DL4J, Nodejs, and Typescript) have a higher percentage (29.94%-48.62%) of respondents/both. A possible explanation is that these four projects are more widely used for development purposes, *e.g.*, Angular is a platform for building mobile and desktop web applications, therefore, such communities appear to be knowledge-sharing and collaborative.

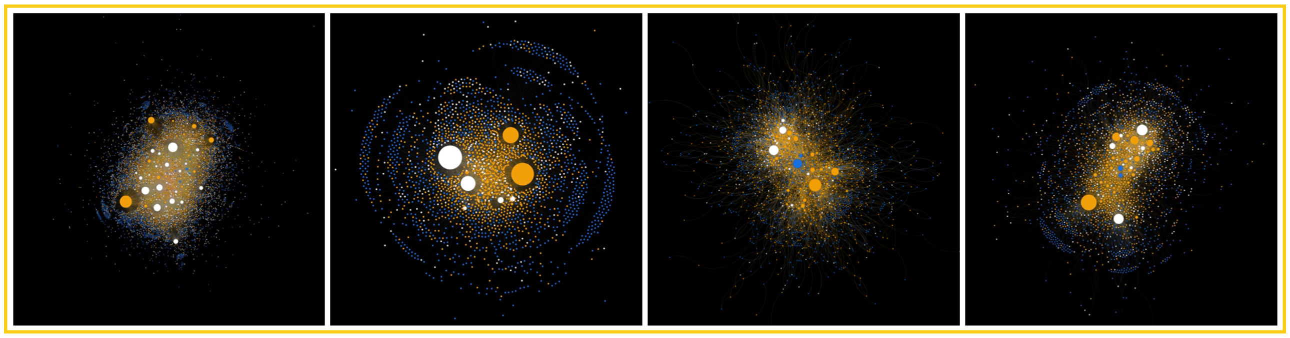
Categorizing developer social networks. Figure 4 shows the social network visualizations of the eight communities generated by Gephi. Each node represents one developer, and the edge denotes the dialog relationship between two developers. We color the vertex of the initiator with blue, the vertex of the respondent with white, and the vertex of both roles with orange. In addition, the node's size indicates its corresponding *degree*. Based on the observation on community structures, we categorize the eight communities into three groups, consisting of: (1) **Polaris network** is a type of highly centralized network where the community is organized around its single focal point; (2) **Constellation network** is a type of moderately centralized network where the community is organized around its multiple focal points; and (3) **Galaxy network** is a type of decentralized network where all individuals in the community have similar relationships. In Figure 4, the four communities on the top (*i.e.*, Angular, DL4J, NodeJS, and Typescript) belong to the Constellation network, *i.e.*, moderately centralized network. Three communities (*i.e.*, Appium, Docker, and Gitter) belong to the Polaris network, *i.e.*, a highly centralized network. The remaining Ethereum community belongs to the Galaxy network, *i.e.*, decentralized network. Previous studies have shown that a highly centralized network may reflect an uneven distribution of knowledge across the community, where knowledge is mostly concentrated at the focal points [38, 44]. Therefore, the three Polaris communities (Appium, Docker, and Gitter) may have a higher risk of single-point failure, if the focal developer is inactive, whereas the Galaxy network (Ethereum) has the lowest risk, followed by the Constellation network (Angular, DL4J, NodeJS, and Typescript).

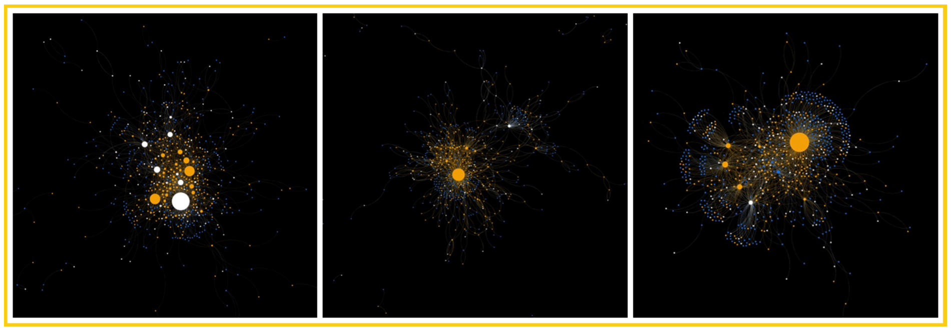
In Table 3, we can also see that the Constellation networks and Polaris networks have higher scores in terms of average degree (1.96-9.15), betweenness (0.000273-0.001342), and closeness (0.31-0.43). The phenomena indicate that the focal points in Constellation networks and Polaris networks make the communities more connected. A study on email-connected social networks [11] shows that the mean betweenness of developers is 0.0114, which on average is higher than live chat communities. Nodes with high betweenness may have considerable influence within a network in allowing information to pass from one part of the network to the other. Lower betweenness indicates that developers in the live chat may have less influence than developers in email in spreading information. However, the average in-degree and out-degree of networks built on emails are significantly lower, with 0.00794 and 0.00666 for developers. While developers in live chat have more concentration and higher density, along with the closeness centrality values, indicating a more closely connected community than that from email. Developers in Constellation communities have higher clustering coefficient scores (0.14-0.60), indicating that developers in Constellation communities are more densely connected, *i.e.*, the developers of Angular, DL4J, Nodejs, and Typescript know each other better than the others.

Answering RQ2: By visualizing the social networks of eight studied communities, we identify three social network structures for developers' live chat. Half of the communities (4/8) are Constellation networks. A minority of the communities (3/8) are Polaris networks. Only one community belongs to the Galaxy network. In

Table 3: Social Network measures of the eight communities

	Constellation				Polaris			Galaxy
	Angular	**DL4J**	**Nodejs**	**Typescript**	**Appium**	**Docker**	**Gitter**	**Ethereum**
Init. %	54.54%	51.38%	70.06%	56.94%	75.04%	76.22%	75.33%	81.70%
Resp. %	13.90%	11.39%	6.83%	12.42%	7.43%	5.84%	9.04%	5.59%
Both %	31.56%	37.23%	23.11%	30.65%	17.53%	17.94%	25.62%	12.71%
Degree	9.15	5.02	2.75	4.2	2.59	1.96	2.07	1.92
Betweenness	0.000273	0.000867	0.000468	0.000786	0.001035	0.001173	0.001342	0.000231
Closeness	0.35	0.42	0.31	0.34	0.35	0.37	0.43	0.34
Diameter	8	6	13	9	10	10	8	15
Clustering coefficient	0.41	0.6	0.14	0.33	0.2	0.13	0.2	0.16


(a) Constellation {from left to right: Angular, DL4J, Nodejs, Typescript }


(b) Polaris {from left to right: Appium, Docker, Gitter }

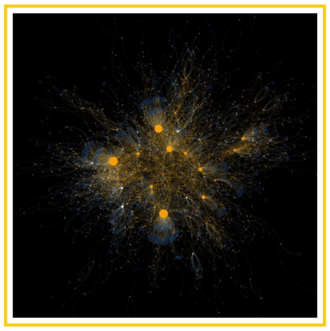

(c) Galaxy { Ethereum }

Figure 4: Visualization of the eight developer live-chat social networks

comparison, we find that developers in the live chat may have less influence than developers in email in spreading information, but have a more closely connected community than that from email.

4.3 RQ3: Discussion Topic

Figure 5 shows the distribution of discussion topics in developer live chat. The figure shows discussion topics in gray and their categories in white, as well as the percentages of the corresponding dialogs. The taxonomy expands outwards from higher-level categories to lower-level categories and topics. In this study, we extend the Beyer *et al.*'s category to accommodate the open and live chat by: (1) adding social chatting and general development categories; and (2) decomposing "Conceptual" and "Discrepancy" categories to distinguish more valuable information such as unwanted behavior and new features. For more information on the dialog topics, we provide a public Github repository with details and examples[4].

[4]https://github.com/LiveChat2021/LiveChat#34-rq3-discussion-topic

The most inner circle shows that, across all eight communities, 89.05% of dialogs are domain-related (DR) topics such as topics related to the business domain of the community, while 10.95% of dialogs are non-domain related (N-DR) topics such as general development or social chatting. DR topics can be further decomposed into three sub-categories based on their different purposes. These include solution-oriented dialogs which have the highest proportion (35.25%), followed by problem-oriented dialogs (32.98%) and knowledge-oriented dialogs (20.83%). Among the 35.25% *solution-oriented* dialogs, 29.37% are about *API usage*, and 5.87% are about *Review*. Among the 32.98% *problem-oriented* dialogs, most of them (20.29%) discuss discrepancy, consisting of *unwanted behavior, do not work, reliability issue, performance issue*, and *test/build failure*. We can see that, developers discuss more 'unwanted behavior' and 'do not work', than reliability issues, performance issues, and test/build failures. Among the 20.83% knowledge-oriented dialogs, most of them (13.75%) discuss conceptual, consisting of *background info,*

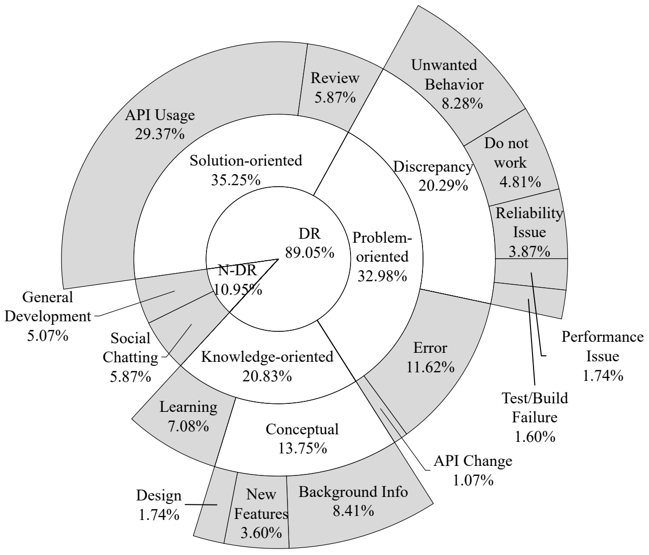


Figure 5: Distribution of discussion topics in developer live chat by reading from center to outside

new features, and *design*. Overall, the top three frequent topics are *API usage (29.37%), Error (11.62%)*, and *Background info (8.41%)*.

Answering RQ3: Developers launch solution-oriented dialogs and problem-oriented dialogs more than knowledge-oriented dialogs. Nearly 1/3 of dialogs are about API usage. Developers discuss more error, unwanted behavior, and do-not-work, than reliability issues, performance issues, and test/build failures.

4.4 RQ4: Interaction Pattern

Interaction patterns. Figure 6 illustrates the six interaction patterns in live chat, constructed using open card sorting as introduced in Section 3.3.4. This figure shows dialog initiators in blue nodes, respondents in yellow nodes. The lines denote the reply-to relationships, and the labels represent developer intents in Table 2. In this work, we identify the following six interaction patterns: (1) **P1: Exploring Solutions**. Given the original questions posted by the dialog initiator, other developers provide possible answers. But

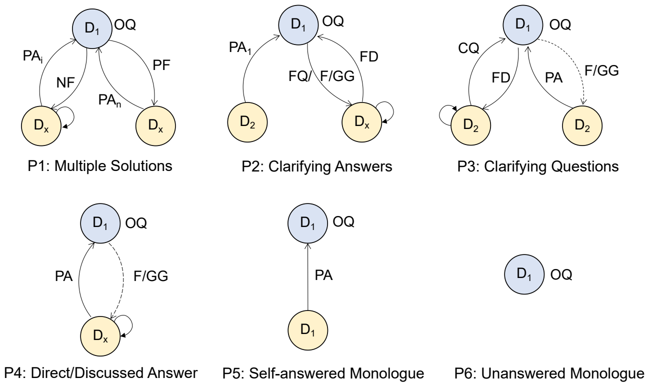


Figure 6: Interactive patterns, F denotes feedback including negative feedback and positive feedback, dashed lines denote optional interaction.

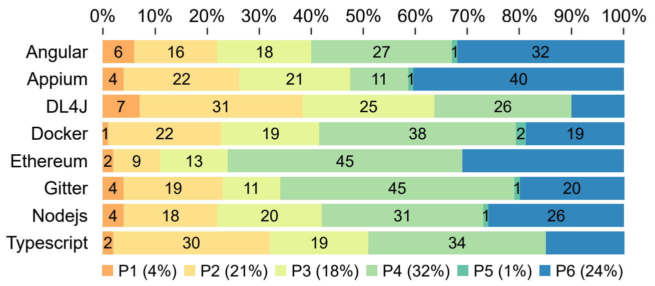


Figure 7: Distribution of interaction patterns among different communities

the initiator gives negative feedback indicating these answers do not address the question. When the correct answer is posted, the initiator gives positive feedback and ends the dialog. (2) **P2: Clarifying Answer**. Given the original questions posted by the dialog initiator, another developer provides a possible answer. Then the initiator posts follow-up questions to clarify the answer until the initiator fully understands. (3) **P3: Clarifying Question.** Given the original questions posted by the dialog initiator, the respondent requires the initiator to clarify the question in more detail until they fully understand. Then the respondent posts an answer, and the initiator gives feedback or greetings. (4) **P4: Direct/Discussed Answer.** Given the original questions posted by the dialog initiator, the respondent directly answers, or answers after an internal discussion. (5) **P5: Self-answered Monologue.** The original questions posted by the dialog initiator are answered by himself or herself. (6) **P6: Unanswered Monologue.** The original questions posted by the dialog initiator are not answered.

Percentage of patterns. Figure 7 shows the percentage of interaction patterns in different communities, and the average percentages are shown in the legends. P1~P6 refer to the six interaction patterns defined above. We can see that the *direct/discussed answer* (P4) pattern takes the largest proportions in most communities. In addition, we note that quite a few dialogs (1%) belong to self-answered monologue, while 24% of dialogs belong to unanswered monologue. Nearly 1/4 of dialogs did not get responses in live chat. We will discuss more monologue in Section 5.1.

Duration of patterns. Figure 8 shows the violin plots with the distribution of duration for each pattern. P1~P5 refer to interaction patterns defined above. Here we only exhibit five patterns because

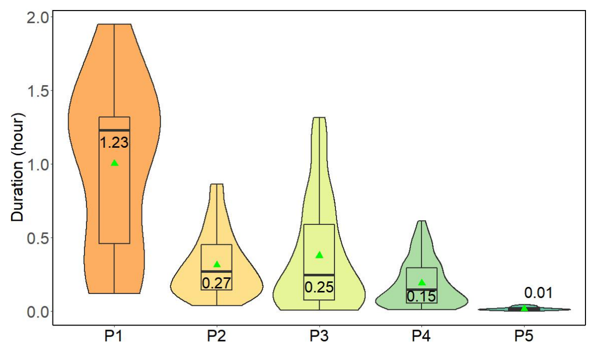


Figure 8: Distribution of duration for interaction patterns, except for unanswered monologues.

the P6 refers to unanswered monologues which barely have a duration. We can see that although P1 takes a small proportion in dialogs, it lasts the longest. Its average duration is 1.00 hours. P2 and P3 last slightly longer than P4. P5 lasts the shortest, and its average duration is 0.02 hour.

Answering RQ4: Six interaction patterns are identified in live chat: *exploring solutions, clarifying answer, clarifying question, direct/discussed answer*, *self-answered monologue*, and *unanswered monologue.* The *direct/discussed answer* pattern takes the largest proportions in most communities. There are still 1/4 dialogs that did not get responses on average. Dialogs that belong to the *Exploring Solutions* pattern last the longest time than others.

5 DISCUSSION

In this work, we take a first look at developers' live chat on Gitter in terms of communication profile, community structure, discussion topic, and interaction pattern. Our work paves the way for other researchers to be able to utilize the same methods in other software communities. Additionally, as communication is a large part of successful software development, and Gitter is one of the main platforms for communication of GitHub users, it is important to explore how software engineers use Gitter and their pain points of using it. Aiming at promoting efficient and effective OSS communications, we discuss the main implications of our findings for OSS developers, communities, platform vendors, and researchers.

5.1 Individual Developers

Based on our findings, we present the following implications for individual OSS developers to attract attentions and receive responses effectively and efficiently.

(1) Provide example code or data when seeking solution help (RQ3, RQ4). In Figure 5, we reported that, there are nearly 1/3 dialogs are problem-oriented. In the live chat, it is important to provide example code in problem-oriented dialogs, to make other developers quickly understand and avoid missing key information. This finding is also in line with the evidence provided by previous studies [12, 14] on Stack Overflow. Here is an example of showing

```
<D1> Hey there, anyone has an idea on this issue: Have quotes
showing up in the UI. I tried to remove it by replacing it
in the .ts file...I'm not sure what else to try.
<D2> I'm having trouble understanding what's going on. Can
you toss your example in a plunker?
```

the importance of using examples in the live chat. The corresponding dialog confirms the Clarifying Question Pattern (P3) with multiple back-and-forth interactions for clarifying questions. If examples are provided at the beginning, the process of issue-resolving would be expedited. Particularly, the Angular community gives great importance to example demonstration. Developers are encouraged to create examples via a demonstration platform, named Plunker [1].

(2) Be aware of low-active hours (RQ1). Our results show that developers are more active during some time slices in live chat. Figure 3(a) demonstrates the most active time slices are UTC 9-10, 13-14, and 18-21, corresponding to Central European/American daytime or Asia nighttime. Noticeably, more developer live chatting happens on Wednesdays and Thursdays than on other weekdays (UTC), which possibly corresponds to communication, coordination, and preparation for integration/release deadlines on Fridays. This observation also confirms the "commercially viable alternative" of the OSS projects reported in recent studies [23, 29, 68]. One of the common findings is that the traditional notion of OSS projects that are driven by voluntary developers is now outdated. OSS has become a commercially viable alternative, and some OSS projects have become critical building blocks for organizations worldwide. For example, Docker is widely used by software companies around the world[5], including Adobe, AT&T, PayPal, *etc.* Therefore, instead of chatting on weekends, developers likely discuss their problems in the live chat on workdays.

While low-active time slices (UTC 1-6) mostly correspond to Central European/American nighttime or Asia day time. In cases where developers find issues and need support during low-active hours, we suggest several options. First, it is recommended to simultaneously post questions to other alternative platforms, *e.g.*, issues and emails. Second, they better follow up in live-chat if not receiving timely responses to their questions posted during low-active hours. Finally, employ some automated reminder bot, for example, to review the list of questions posted during low-active hours.

(3) Avoid asking amid ongoing discussions (RQ4). When identifying the unanswered monologues patterns from dialogs, we note that 30% of them are launched in the middle of ongoing active and intense conversations on a different topic. In such cases, new questions are easily flooded by the utterances of the ongoing discussions. Therefore, to increase the opportunity of getting responses, developers could post their questions after the ongoing discussions. In case that an urgent matter emerges, we suggest that the platform vendors provide special accommodations to flag such urgency and redirect the team's attention to it, such as multi-threaded conversation (e.g., in Slack) or a highlight tag for urgent questions and let others supervise the usage of an urgent tag to avoid abuse.

5.2 OSS Communities

we provide the following recommendations for OSS managers to improve the management and coordination of the communities.

(1) Mitigate the risk of single-point failure (RQ2). As reported in RQ2, the three Polaris communities (Appium, Docker, and Gitter) may have a higher risk of single-point failure, if the focal developer left or became inactive. It is noticeable that there are some second focal points smaller than those most focal ones in the three Polaris networks, and this may suggest practical strategies in order to mitigate the risk of single-point failure. For example, the Polaris communities may design and employ appropriate incentives or policies to second focal developers, for improving the resilience of the live chat communities.

(2) Improve OSS documentation for newcomers (RQ1, RQ4). It is reported that some newcomers complained that it is hard to start on a new project and get timely help from other community members, which may make them gradually lose motivation, or even give up on contributing [63]. To facilitate newcomers to familiarize themselves and make contributions in a more efficient manner, OSS communities may consider utilizing the results of our study to improve OSS documentation. For example, the results of RQ1 show active and low-active time slices, and the results of RQ4 show many

[5]https://www.docker.com/customers

unanswered monologues are asking amid ongoing discussions. That information could be incorporated into the README documents for newcomers, who are looking to contribute to a project and how to get timely help from others.

5.3 Platform Vendors

This section discusses several desired features for facilitating more productive conversations. Specifically, these are organized from communication platform vendors' perspective, in support of more intelligent and productive chatting options, leveraging on mining and knowledge sharing of intensive historical conversations.

(1) Highlight and organize conversation topics (RQ3). As suggested by the previous study, multi-dimensional separation of concerns [47] is a powerful concept supporting collaborative development by breaking a large discussion down into many smaller units. It highlights that the online communication platform vendors could provide support for a set of predefined panels that focus on certain topics. In the results of RQ3, we provide a taxonomy of discussion topics in live chat. The online communication platform vendors could refer to this taxonomy to create topic panels. For example, API-usage panel, Error panel, Background-info panel, *etc.* These multiple panels could bring the following benefits to community members: (i) quickly understanding and retrieving the intents of the dialog initiators; (ii) reducing interference and focusing more on topics of interest; and (iii) identifying important information reported by the developers.

(2) Annotate important questions (RQ3). When investigating dialog topics, we note that certain types of dialogs suggest information for future software evolution. For example, there are 3.6% dialogs discussing new features, and 8.28% discussing unwanted behaviors. These dialogs are valuable for product teams to plan future releases. In the meanwhile, other types of dialogs indicate unrevealed defects of existing systems. For example, 11.62% dialogs discuss errors, 4.81% discussing something that does not work, 3.87% discussing reliability issues, and 1.74% discussing performance issues. Properly annotating those dialogs with "Feature Request", "Enhancement", and "Bugs" would help to preserve valuable information, and contribute to productivity and quality improvement of the software. As an example, techniques for mining live chat have been explored for identifying feature requests from chat logs [57].

5.4 Researchers

Research in the SE area could dedicate to promote efficient and effective OSS communication in the following directions.

(1) Automatically recommend similar questions (RQ4). Existing online communication platforms only record the massive history chat messages, but do not consider a deeper utilization of those historical data. Actually, we note that developers post similar questions in live chat sometimes. In DL4J, one initiator posted a question, and he got a reply like this: "Someone else asked a very similar question a while ago." However, it is not easy for the initiator to accurately retrieve the similar question out from the massive history messages. In addition, some questions that got unanswered are largely due to many similar questions being previously answered. Therefore, we consider that, it will save developers' effort if researchers could develop approaches that automatically recommend similar questions and the corresponding discussions.

(2) Automatically assign appropriate respondents (RQ4). By analyzing the dialogs belonging to exploring solution patterns (P1), we note that respondents who are not quite familiar with the technologies related to the posted questions might give ineffective solutions. Although such discussions could make developers understand the problem better, the multiple fail-and-try interactions still prolong the process of issue-resolving. Therefore, to make conversations more productive, it is expected to develop approaches that could recommend or assign appropriate respondents according to their historical answers.

(3) Automatically push valuable information to project repositories (RQ3). Valuable information such as feature requests or issue reports, either manually annotated by developers or automatically detected by tools, needs to be well-documented and well-traced in the scope of project repositories. Typically, code repositories such as Github or Gitlab provide the functionality of issue tracking. It would be more efficient if researchers could provide a convenient way to directly push or integrate the valuable information into the code repository. In addition, the following linguistic patterns might be helpful to automatically classify dialog topics. It is observed that questions from the *API usage* category include phrases such as: "how to do sth?", "how can I do sth?", "can anyone help me with sth?", "is there any way to sth?", or "I want/need to do sth". For example, "How to bundle my Angular 2 app into a 'bundle.js' file?" and "Can anyone help me with PWA using angular 2". Questions from the *Error* category are likely to be "I get/receive this error", "Anyone had an error like this", "Does anyone know a solution for sth", or directly posting the specific exception names. For example, "I receive a NotFound error" and "Anyone had an error like this before when trying to load a route?" Questions from the *Background info* category are likely to be "what/why/when...", "I would like to know sth", or "is there sth for...". For example, "When Appium will support Xcode 8.2?" and "Are there any limitations for automating the iOS app made with Swift 3?".

(4) Analyze effects of social chatting (RQ3). As reported in RQ3, 10.95% of dialogs are non-domain related (N-DR) topics such as general development or social chatting. A recent study [46] emphasizes an important role of social interactions, such as the simple phrase "How was your weekend?", to show peer support for developers working at home during the COVID-19 pandemic. Future work may explore more patterns and effects of social chatting, *e.g.*, pre/post-pandemic comparison.

5.5 Threats To Validity

External Validity. The external threats relate to the generalizability of the proposed approach. Our empirical study used eight Top-1 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.

Internal Validity. The internal threats relate to experimental errors and biases. The first threat relates to the accuracy of the dialog disentanglement model adopted by us. Although we select the best model to disentangle dialogs from the state-of-the-art approaches, the accuracy score for the best model is still not quite satisfactory. It will have an impact on the results of RQ1 and RQ2.

To address this issue, one of our ongoing works is to build a new efficient dialog disentanglement model based on deep learning to improve the accuracy of existing disentanglement approaches. The second threat relates to the random sampling process. Sampling may lead to incomplete results, *e.g.*, topic taxonomy and interaction patterns. In the future, we plan to enlarge the analyzed dataset and inspect whether new topics or interaction patterns are emerging. The third threat might come from the process of manual disentanglement and card sorting. We understand that such a process is subject to introducing mistakes. To reduce that threat, we establish a labeling team, and perform 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, manual labeling of topics and interactive patterns is a construct threat. To minimize this threat, we use a well-known approach used by previous work [5, 10, 53] to build reasonable taxonomies for textual software artifacts.

6 RELATED WORK

Our work is related to previous studies that focused on synchronous and asynchronous communication in the OSS community.

Synchronous Communication in OSS community. Recently, more and more work has realized that live chat via modern communication platforms plays an increasingly important role in team communication. Lin *et al.* [40] conducted an exploratory study on understanding the role of Slack in supporting software engineering by surveying 104 developers. Their research revealed that developers use Slack for personal, team-wide, and community-wide purposes, and developers use bots for team and task management in their daily lives. They highlighted that live chat plays an increasingly significant role in software development, replacing email in some cases. Shihab *et al.* [58, 59] 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 open source developers, and highlighted the wealth of information that can be obtained from developer chat messages. Yu *et al.* [67] analyzed the usage of two communication mechanisms in global software development projects, which are synchronous (IRC) and asynchronous (mailing list). Their results showed that developers actively use both communication mechanisms in a complementary way. To sum up, existing empirical analysis of live chat mainly focused on the usage purpose [40], the usage of live meetings [58, 59], and comparison with different communication mechanisms and knowledge-share platforms [67]. There is a lack of in-depth analysis of the community properties and the detailed discussion contents. Our study bridges that gap with a large-scale analysis of communication profiles, community structures, dialog topics, and interaction patterns in live chat.

Asynchronous Communication in OSS community. Prior studies have empirically analyzed asynchronous communication in the OSS community, including mailing-list, issue discussions, and Stack Overflow. Bird *et al.* [11] mined email social network on the Apache HTTP server project. They reported that the email social network is a typical electronic community: a few members account for the bulk of the messages sent, and the bulk of the replies. Di Sorbo *et al.* [60] 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 [48], Huang *et al.* [36] 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. Arya *et al.* [4] identified 16 information types, such as new issues and requests, solution usage, *etc.*, through quantitative content analysis of 15 issue discussion threads in Github. They also provided a supervised classification solution by using Random Forest with 14 conversational features to classify sentences. Allamanis and Sutton [3] 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". Similarly, Rosen and Shihab [51] employed Latent Dirichlet Allocation-based topic models to help us summarize the mobile-related questions from Stack Overflow. Our work differs from existing research in that we focus on synchronous communication which poses different challenges as live chat logs are informal, unstructured, noisy, and interleaved.

7 CONCLUSION AND FUTURE WORK

In this paper, we have presented the first large-scale study to gain an empirical understanding of OSS developers' live chat. Based on 173,278 dialogs taken from eight popular communities on Gitter, we explore the temporal communication profiles of developers, the social networks and their properties towards the community, the taxonomy of discussion topics, and the interaction patterns in live chat. Our study reveals a number of interesting findings. Moreover, we provide recommendations for both OSS developers and communities, highlight advanced features for online communication platform vendors, and provoke insightful future research questions for OSS researchers. In the future, we plan to investigate how well can we automatically classify the dialogs into different topics, as well as attempt to construct knowledge bases according to already answered questions and their corresponding solutions from live chat. We hope that the findings and insights that we have uncovered will pave the way for other researches, help drive a more in-depth understanding of OSS development collaboration, and promote a better utilization and mining of knowledge embedded in the massive chat history. To facilitate replications or other types of future work, we provide the utterance data and disentangled dialogs used in this study online: https://github.com/LiveChat2021/LiveChat.

ACKNOWLEDGMENTS

We deeply appreciate anonymous reviewers for their constructive and insightful suggestions towards improving this manuscript. This work is supported by the National Key Research and Development Program of China under Grant No. 2018YFB1403400, the National Science Foundation of China under Grant No. 61802374, 62002348, and 62072442, and Youth Innovation Promotion Association CAS.

REFERENCES

[1] A Tool to Prototype and Experiment Angular Codes. 2020. Plnkr. http://plnkr.co/.

[2] Explosion AI. 2020. Spacy. https://spacy.io/.

[3] Miltiadis Allamanis and Charles Sutton. 2013. 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, 53–56. https://doi.org/10.1109/MSR.2013.6624004

[4] Deeksha Arya, Wenting Wang, Jin L. C. Guo, and Jinghui Cheng. 2019. Analysis and Detection of Information Types of Open Source Software Issue Discussions. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019*. 454–464. https://doi.org/10.1109/ICSE.2019.00058

[5] Mehdi Bagherzadeh and Raffi Khatchadourian. 2019. Going Big: A Large-scale Study on What Big Data Developers Ask. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019*. ACM, 432–442. https://doi.org/10.1145/3338906.3338939

[6] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 3.

[7] Alex Bavelas. 1950. Communication Patterns in Task-oriented Groups. *The journal of the acoustical society of America* 22, 6 (1950), 725–730.

[8] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger. 2017. *Analyzing the Relationships between Android API Classes and Their References on Stack Overflow*. Technical Report. University of Klagenfurt, University of Sannio.

[9] Stefanie Beyer, Christian Macho, Martin Pinzger, and Massimiliano Di Penta. 2018. Automatically Classifying Posts into Question Categories on Stack Overflow. In *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018*. ACM, 211–221. https://doi.org/10.1145/3196321.3196333

[10] Stefanie Beyer and Martin Pinzger. 2014. A Manual Categorization of Android App Development Issues on Stack Overflow. In *30th IEEE International Conference on Software Maintenance and Evolution*. IEEE Computer Society, 531–535. https://doi.org/10.1109/ICSME.2014.88

[11] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining Email Social Networks. In *Proceedings of the 2006 international workshop on Mining software repositories*. 137–143. https://doi.org/10.1145/1137983.1138016

[12] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. 2013. Building Reputation in StackOverflow: An Empirical Investigation. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 89–92. https://doi.org/10.1109/MSR.2013.6624013

[13] Jérémie Bouttier, Philippe Di Francesco, and Emmanuel Guitter. 2003. Geodesic Distance in Planar Graphs. *Nuclear physics B* 663, 3 (2003), 535–567.

[14] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to Ask for Technical Help? Evidence-based Guidelines for Writing Questions on Stack Overflow. *Inf. Softw. Technol.* 94 (2018), 186–207. https://doi.org/10.1016/j.infsof.2017.10.009

[15] Preetha Chatterjee, Kostadin Damevski, Nicholas A. Kraft, and Lori L. Pollock. 2020. Software-related Slack Chats with Disentangled Conversations. In *MSR '20: 17th International Conference on Mining Software Repositories*. ACM, 588–592. https://doi.org/10.1145/3379597.3387493

[16] Microsoft Corporation. 2020. Nodejs. https://nodejs.org/en/.

[17] Microsoft Corporation. 2020. Typescript. https://www.typescriptlang.org/.

[18] Fabio Crestani and Mounia Lalmas. 2001. Logic and Uncertainty in Information Retrieval. In *Lectures in Information Retrieval, Lecture Notes in Computer Science*. Springer Verlag. https://doi.org/10.1007/3-540-45368-7_9

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[20] Reinhard Diestel. 2005. Graph theory. 2005. *Grad. Texts in Math* 101 (2005).

[21] Inc Docker. 2020. Docker. https://www.docker.com/.

[22] Sally Fincher and Josh D. Tenenberg. 2005. Making Sense of Card Sorting Data. *Expert Syst. J. Knowl. Eng.* 22, 3 (2005), 89–93. https://doi.org/10.1111/j.1468-0394.2005.00299.x

[23] Brian Fitzgerald. 2006. The Transformation of Open Source Software. *MIS Q.* 30, 3 (2006), 587–598.

[24] Ethereum Foundation. 2020. Ethereum. https://ethereum.org/en/.

[25] JS Foundation. 2020. Appium. http://appium.io/.

[26] Linton C Freeman. 1977. A Set of Measures of Centrality Based on Betweenness. *Sociometry* (1977), 35–41.

[27] Adam Gibson. 2020. Deeplearning4j. https://deeplearning4j.org/.

[28] Gitter. 2020. REST API. https://developer.gitter.im/docs/rest-api.

[29] Jesús M. González-Barahona and Gregorio Robles. 2013. Trends in Free, Libre, Open Source Software Communities: From Volunteers to Companies / Aktuelle Trends in Free-, Libre-, und Open-Source-Software-Gemeinschaften: Von Freiwilligen zu Unternehmen. *it Inf. Technol.* 55, 5 (2013), 173–180. https://doi.org/10.1524/itit.2013.1012

[30] Google. 2020. Angular. https://angular.io/.

[31] Google. 2020. Gitter. https://gitter.im/.

[32] Google. 2020. Slack. https://slack.com/.

[33] Gaoyang Guo, Chaokun Wang, Jun Chen, and Pengcheng Ge. 2018. Who Is Answering to Whom? Finding "Reply-To" Relations in Group Chats with Long Short-Term Memory Networks. In *Proceedings of the 7th International Conference on Emerging Databases*. https://doi.org/10.1007/s10586-018-2031-4

[34] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. 2013. Communication in Open Source Software Development Mailing Lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13*. IEEE Computer Society, 277–286. https://doi.org/10.1109/MSR.2013.6624039

[35] Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. 2020. What Do Programmers Discuss About Deep Learning Frameworks. *Empir. Softw. Eng.* 25, 4 (2020), 2694–2747. https://doi.org/10.1007/s10664-020-09819-6

[36] Qiao Huang, Xin Xia, David Lo, and Gail C. Murphy. 2018. Automating Intention Mining. *IEEE Transactions on Software Engineering* PP, 99 (2018), 1–1. https://doi.org/10.1109/ICSM.2015.7332474

[37] Verena Käfer, Ivan Bogicevic, Stefan Wagner, and Jasmin Ramadani. 2018. Communication in Open-source Projects-end of the E-mail Era?. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018*. ACM, 242–243. https://doi.org/10.1145/3183440.3194951

[38] Valdis Krebs and June Holley. 2004. Building Sustainable Communities through Social Network Development. *The Nonprofit Quarterly* 11 (01 2004), 46–53.

[39] Jonathan K. Kummerfeld, Sai R. Gouravajhala, Joseph Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros Polymenakos, and Walter S. Lasecki. 2019. A Large-scale Corpus for Conversation Disentanglement. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. https://doi.org/10.18653/v1/p19-1374

[40] Bin Lin, Alexey Zagalsky, Margaret-Anne D. Storey, and Alexander Serebrenik. 2016. 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*. 333–336. https://doi.org/10.1145/2818052.2869117

[41] Hui Liu, Zhan Shi, Jia-Chen Gu, Quan Liu, Si Wei, and Xiaodan Zhu. 2020. End-to-End Transition-based Online Dialogue Disentanglement. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. ijcai.org, 3868–3874. https://doi.org/10.24963/ijcai.2020/535

[42] Troupe Technology Ltd. 2020. Gitter. https://gitter.im/.

[43] Yao Lu, Xinjun Mao, Minghui Zhou, Yang Zhang, Tao Wang, and Zude Li. 2020. Haste Makes Waste: An Empirical Study of Fast Answers in Stack Overflow. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020*. IEEE, 23–34. https://doi.org/10.1109/ICSME46990.2020.00013

[44] Manju, K., Ahuja, Kathleen, M., and Carley. 1998. Network Structure in Virtual Organizations. *Journal of Computer Mediated Communication* (1998). https://doi.org/10.1111/j.1083-6101.1998.tb00079.x

[45] Andrew Meneely, Laurie A. Williams, Will Snipes, and Jason A. Osborne. 2008. Predicting Failures with Developer Networks and Social Network Analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008*. ACM, 13–23. https://doi.org/10.1145/1453101.1453106

[46] Courtney Miller, Paige Rodeghero, Margaret-Anne D. Storey, Denae Ford, and Thomas Zimmermann. 2021. "How Was Your Weekend?" Software Development Teams Working From Home During COVID-19. *CoRR* abs/2101.05877 (2021).

[47] Ana Moreira, Awais Rashid, and João Araújo. 2005. Multi-Dimensional Separation of Concerns in Requirements Engineering. In *13th IEEE International Conference on Requirements Engineering (RE 2005)*. IEEE Computer Society, 285–296. https://doi.org/10.1109/RE.2005.46

[48] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C Gall. 2015. How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution. (2015), 281–290.

[49] Esteban Parra, Ashley Ellis, and Sonia Haiduc. 2020. GitterCom: A Dataset of Open Source Developer Communications in Gitter. In *MSR '20: 17th International Conference on Mining Software Repositories*. ACM, 563–567. https://doi.org/10.1145/3379597.3387494

[50] Chen Qu, Liu Yang, W. Bruce Croft, Yongfeng Zhang, Johanne R. Trippas, and Minghui Qiu. 2019. User Intent Prediction in Information-seeking Conversations. In *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval, CHIIR 2019*. ACM, 25–33. https://doi.org/10.1145/3295750.3298924

[51] Christoffer Rosen and Emad Shihab. 2016. What Are Mobile Developers Asking About? A Large Scale Study Using Stack Overflow. *Empir. Softw. Eng.* 21, 3 (2016), 1192–1223. https://doi.org/10.1007/s10664-015-9379-3

[52] Gordon Rugg and Peter McGeorge. 2005. The Sorting Techniques: A Tutorial Paper on Card Sorts, Picture Sorts and Item Sorts. *Expert Syst. J. Knowl. Eng.* 22, 3 (2005), 94–107. https://doi.org/10.1111/j.1468-0394.2005.00300.x

[53] Khadijah Al Safwan and Francisco Servant. 2019. Decomposing the Rationale of Code Commits: The Software Developer's Perspective. In *Proceedings of the ACM*

Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019. ACM, 397–408. https://doi.org/10.1145/3338906.3338979

[54] Jorge M. Santos and Mark J. Embrechts. 2009. On the Use of the Adjusted Rand Index as a Metric for Evaluating Supervised Classification. In *Artificial Neural Networks - ICANN 2009, 19th International Conference, Limassol (Lecture Notes in Computer Science, Vol. 5769)*. Springer, 175–184. https://doi.org/10.1007/978-3-642-04277-5_18

[55] Roland Robert Schreiber and Matthäus Paul Zylka. 2020. Social Network Analysis in Software Development Projects: A Systematic Literature Review. *Int. J. Softw. Eng. Knowl. Eng.* 30, 3 (2020), 321–362. https://doi.org/10.1142/s021819402050014x

[56] Dou Shen, Qiang Yang, Jian-Tao Sun, and Zheng Chen. 2006. Thread Detection in Dynamic Text Message Streams. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 35–42. https://doi.org/10.1145/1148170.1148180

[57] Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, and Qing Wang. 2020. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network. In *ICSE '20: 42nd International Conference on Software Engineering*. ACM, 641–653. https://doi.org/10.1145/3377811.3380356

[58] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. 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)*. 107–110. https://doi.org/10.1109/MSR.2009.5069488

[59] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. Studying the Use of Developer IRC Meetings in Open Source Projects. In *25th IEEE International Conference on Software Maintenance (ICSM 2009)*. 147–156. https://doi.org/10.1109/ICSM.2009.5306333

[60] Andrea Di Sorbo, Sebastiano Panichella, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2015. Development Emails Content Analyzer: Intention Mining in Developer Discussions (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*. 12–23. https://doi.org/10.1109/ASE.2015.12

[61] Alexander Strehl and Joydeep Ghosh. 2002. Cluster Ensembles — A Knowledge Reuse Framework for Combining Multiple Partitions. *J. Mach. Learn. Res.* 3 (2002), 583–617.

[62] Neny Sulistianingsih and Edi Winarko. [n.d.]. Performance Analysis of Molecular Complex Detection in Social Network Datasets. *International Journal of Computer Applications* 975 ([n. d.]), 8887.

[63] Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A first look at good first issues on GitHub. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 398–409. https://doi.org/10.1145/3368089.3409746

[64] Christoph Treude, Ohad Barzilay, and Margaret-Anne D. Storey. 2011. How Do Programmers Ask and Answer Questions on the Web?. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011*. ACM, 804–807. https://doi.org/10.1145/1985793.1985907

[65] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press. https://doi.org/10.1017/CBO9780511815478

[66] Duncan J Watts and Steven H Strogatz. 1998. Collective Dynamics of 'Small-world' Networks. *nature* 393, 6684 (1998), 440–442.

[67] Liguo Yu, Srini Ramaswamy, Alok Mishra, and Deepti Mishra. 2011. 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*. 218–227. https://doi.org/10.1007/978-3-642-25126-9_32

[68] Yuxia Zhang, Minghui Zhou, Klaas-Jan Stol, Jianyu Wu, and Zhi Jin. 2020. How Do Companies Collaborate in Open Source Ecosystems?: An Empirical Study of OpenStack. In *ICSE '20: 42nd International Conference on Software Engineering*. ACM, 1196–1208. https://doi.org/10.1145/3377811.3380376