How are issue reports discussed in Gitter chat rooms?

Hareem Sahar[1], Abram Hindle[1], Cor-Paul Bezemer[2]

University of Alberta

Edmonton, Canada

Abstract

Informal communication channels like mailing lists, IRC and instant messaging play a vital role in open source software development by facilitating communication within geographically diverse project teams e.g., to discuss issue reports to facilitate the bug-fixing process. More recently, chat systems like Slack and Gitter have gained a lot of popularity and developers are rapidly adopting them. Gitter is a chat system that is specifically designed to address the needs of GitHub users. Gitter hosts project-based asynchronous chats which foster frequent project discussions among participants. Developer discussions contain a wealth of information such as the rationale behind decisions made during the evolution of a project. In this study, we explore 24 open source project chat rooms that are hosted on Gitter, containing a total of 3,133,106 messages and 14,096 issue references. We manually analyze the contents of chat room discussions around 457 issue reports. The results of our study show the prevalence of issue discussions on Gitter, and that the discussed issue reports have a longer resolution time than the issue reports that are never brought on Gitter.

Keywords:

developer discussions, Gitter, issue reports

Email address: `hareeme@ualberta.ca` (Hareem Sahar)

[1]Department of Computing Science, University of Alberta, Canada

[2]Analytics of Software, Games and Repository Data (ASGAARD) lab, University of Alberta, Canada

1. Introduction

Open source software (OSS) development uses the expertise of developers from all over the world, who communicate with each other via email, mailing lists [1], IRC channels [2], and modern communication platforms like Gitter and Slack [3]. Moreover, people interested in improving their programming skills connect through the aforementioned informal communication channels [4] and seek help from project developers. The vast amount of knowledge that is available on these platforms has been exploited by researchers to improve API learning [5][6], source code documentation [7], and automatic generation of source code comments [8]. In addition, the developer discussions on Stack Overflow have been leveraged for more sophisticated tasks, such as supporting recommendations in the IDEs [9][10][11], bug triaging [12], and in building the thesauri and knowledge graphs of software-specific terms [13]. These successes suggest that developer communications in unexplored platforms may also contain potentially useful knowledge.

There has been a recent shift from the traditional communication platforms, such as emails and mailing lists, to modern team-and-project oriented instant messaging systems like Gitter, Slack, and IRC [14]. These systems not only keep teams connected to each other by enabling real-time text communication among groups, but also provide a tight integration with various external services. Additionally, software development teams prefer these chat systems because of their more contained environment which makes it easier for the members to closely interact with each other [3]. However, despite, their wide adoption, little is known about how project teams use these platforms during project development and management.

Prior work studied Slack [15] [3] and IRC [2] [16] but, in this paper, we explore an increasingly popular [17] project-oriented chat system, Gitter, using a combination of qualitative and quantitative analysis methods. The success of Gitter could be attributed to the fact that it was created to address the needs of GitHub users, and that it is an open-source chat system as opposed to the proprietary platforms like Slack. In addition, Gitter chat archives are publicly available and can be easily mined using a REST API,[3] opening up new avenues of research for the Mining Software Repositories community. Apart from content archiving, Gitter provides several ready-to-go integrations with GitHub repositories and bots. Pull requests, issue report linking, and GitHub flavored mark-down are also supported, making it easier for the developers to directly talk about project artifacts or releases in the Gitter chat rooms.

Our study is the first to empirically investigate the role of Gitter in supporting software engineering teams. Our goal is to assess the impact of Gitter on project and team dynamics. In this paper, we focus on *issue report* discussions in Gitter chat rooms. Issue reports are used to keep track of bugs, request enhancements, and to document software code. The reason of limiting our analysis to issue report discussions is that issues are an integral part of the bug-fixing process of a software project, and play a crucial role in a project's success. We anticipate that developers discuss issue reports throughout a project's evolution to better understand the problems in their project and to design strategies to resolve these problems. Therefore, focusing on issue report discussions allows us to analyze the impact of Gitter on various project-related activities during the evolution of a software project. We address the following research questions (RQs) in our study.

RQ1: Who refers to and discusses issue reports in Gitter chat rooms?

More than 50% issue reports were referenced by the end-users in 14 out of 24 chat rooms.

RQ2: What is discussed about issue reports on Gitter?

Our manual analysis shows that the main purpose of issue report references is to ask for technical support or to discuss issue-related activity happening within the GitHub issue trackers. On the other hand, issue reports from other projects are also mentioned in the chat rooms, as some projects share common

[3] https://developer.gitter.im/docs/rest-api

dependencies and are affected by language features and library dependencies in a similar way.

RQ3: How does the issue resolution time of issues that are discussed and not discussed on Gitter differ?

We speculated that discussing an issue report on Gitter would cause the issue to resolve faster. To our surprise, we found that issue reports that are discussed on Gitter have a significantly higher resolution time as compared to issue reports that are not discussed on Gitter.

RQ4: Does the discussion on Gitter impact the issue's activity and resolution time of the issue?

We found an increase in the number of issue comments in the GitHub issue tracker after an issue report was referenced on Gitter. Our preliminary result suggests that long-standing issues may get resolved when brought to Gitter.

The remainder of this paper is structured as follows: Section 2 gives a brief introduction of Gitter. Section 3 presents prior work related to our paper. Section 4 explains our data extraction and processing methodology. In Section 5, we present the findings of the research questions followed by a discussion in Section 6. Section 7 explains the threats to the validity of our study and Section 8 concludes the paper.

2. Gitter

Gitter is an open source GitHub-based chat system with around 90K communities, 300K chat rooms and 800K people who are mostly developers and GitHub users. Each Gitter community contains multiple chat rooms, which could be general-purpose, organizational or repository chat rooms. The repository chat rooms are linked to the GitHub project repositories. Initially, only repository chat rooms existed but Gitter changed its structure in August 2016 after which chat rooms and communities that are not directly related to objects in GitHub could be created. Gitter chat rooms are mostly public as opposed to the closed nature of Slack chat rooms. Being public means, anyone interested in the project can join in and have a conversation with project maintainers. There is also no room size limit which is why it perfectly suits the needs of large teams.[4]

Similar to Slack, Gitter supports a plethora of external integrations[5] and services, such as Gitlab, GitHub, Travis, Jenkins, Heroku, and, Trello. However, Gitter's integrations with external services are ready-to-go while Slack integrations have to be installed. Furthermore, a fully searchable history is offered, as developers can browse activity archives dating back to when the chat room was created, something that is limited by Slack.[6] Finally, one can connect to Gitter using an IRC client. Due to these factors, Gitter has rapidly gained the interest of software development teams, and many projects on GitHub now have a Gitter badge, confirming its wide adoption and making it the top developer communication channel after the issue trackers and mailing lists [17]. Our study is the first to assess the role and impact of Gitter in open source software development.

3. Related Work

In this section, we present studies that analyze developer communications and their impact on a software project.

Software developers use online chat services such as Slack [3] [15], IRC [2], Hipchat, Gitter [14] and Microsoft Teams to communicate about project tasks, to learn new programming languages and technologies or to ask project specific questions. Most studies on developer chats focused on learning about how development teams use chat communities or to analyze developers' behaviours and interactions [2] [18].

Shihab et al. [2] explored IRC channels of the GTK and the GNOME project.

---

[4]https://www.freecodecamp.org/news/so-yeah-we-tried-slack-and-we-deeply-regretted-it-391bcc714c81/

[5]https://gitlab.com/gitlab-org/gitter/webapp/blob/master/docs/integrations.md

[6]https://slack.com/intl/en-ca/pricing

Their focus was on the IRC meeting participants, the contents of their discussions, and the style in which the meetings are run. Panichella et al. [1] compared how developer collaboration links differ across 3 different kinds of communication channels, including mailing lists, issue trackers, and IRC chat logs. A similar study was conducted by Yu et al. [19] to find out how developers of a small open source project use real-time IRC and asynchronous email communications in Global Software Development.

Other work has focused on mining and extracting specific types of information from developer communications. For instance, Bird et al. [20] mined email social networks of the Apache HTTP server project and discovered that the developer mailing list and source code activity are correlated. Alkadhi et al. [18] pointed out the presence of rationale in developer chats and examined the frequency and completeness of available rationale in HipChat and IRC messages. They also studied the potential of automatic machine learning techniques for rationale extraction from chat messages whereas others employed machine learning techniques to filter off-topic discussions from IRC chats [16]. Lately, Slack and Gitter have gained a lot of popularity and therefore researchers have also started to explore the potential of information available in these platforms [21] for improving software engineering processes and tools. Lin et al. [3] assessed the impact of the use of Slack on the software development teams and processes by directly interviewing the developers. They found that Slack enables new ways to collaborate and supports developers by serving a wide range of purposes. Chatterjee et al. [15] demonstrated the potential usefulness of Slack data by comparing and contrasting the characteristics and the contents of Slack chat data with the famous Stack Overflow data [22].

Gitter is a modern project-oriented chat system [14], that became popular as an open-source alternative to Slack. To the best of our knowledge, no prior work explored Gitter to see what kind of project evolution-relevant information is available in Gitter chat rooms. This is the first study to analyze developer discussions on Gitter and their link to the project issue reports.

Researchers have also investigated the role of communication between developers in coordinating development and maintenance activities in software projects, such as requirements understanding, making design decisions and bug injection in source code. Bettenburg and Hassan [23] examined the effect of social interactions between developers on the software quality. Abreu and Premraj [24] showed that the frequency of communication among the project's entire development community has a relation with the number of bug introducing changes. In another study, Wolf et al. [25] exploited social network analysis measures obtained from communication among developers to predict build failures. Similarly, Bacchelli et al. [26] used code popularity metrics obtained from email communication among developers for bug prediction while Sajedi et al. [12] leverage the developer communications in Q&A platforms as source of information for bug triaging.

Several studies have also examined the effect of human communication and coordination factors on the issue resolution time in projects. Ortu et al. [27] found out that politeness has an effect on the issue resolution time in OSS. Yu et al. [28] suggested that discussions throughout the bug fixing process are important to clarify the reported problem and reach a solution. They also found evidence of the association between discussions and bug reworking. To this end, we explore issue discussions in Gitter chat rooms and investigate the issue resolution time of issue reports discussed in chats.

4. Mining Gitter Data

In this section, we explain our data extraction and parsing methodology which is outlined in Figure 1. The dataset and scripts are also available on GitHub.[7]

4.1. Selecting Chat Rooms

In this study, we focus on chat rooms:

[7] https://github.com/Hareem-E-Sahar/gitter

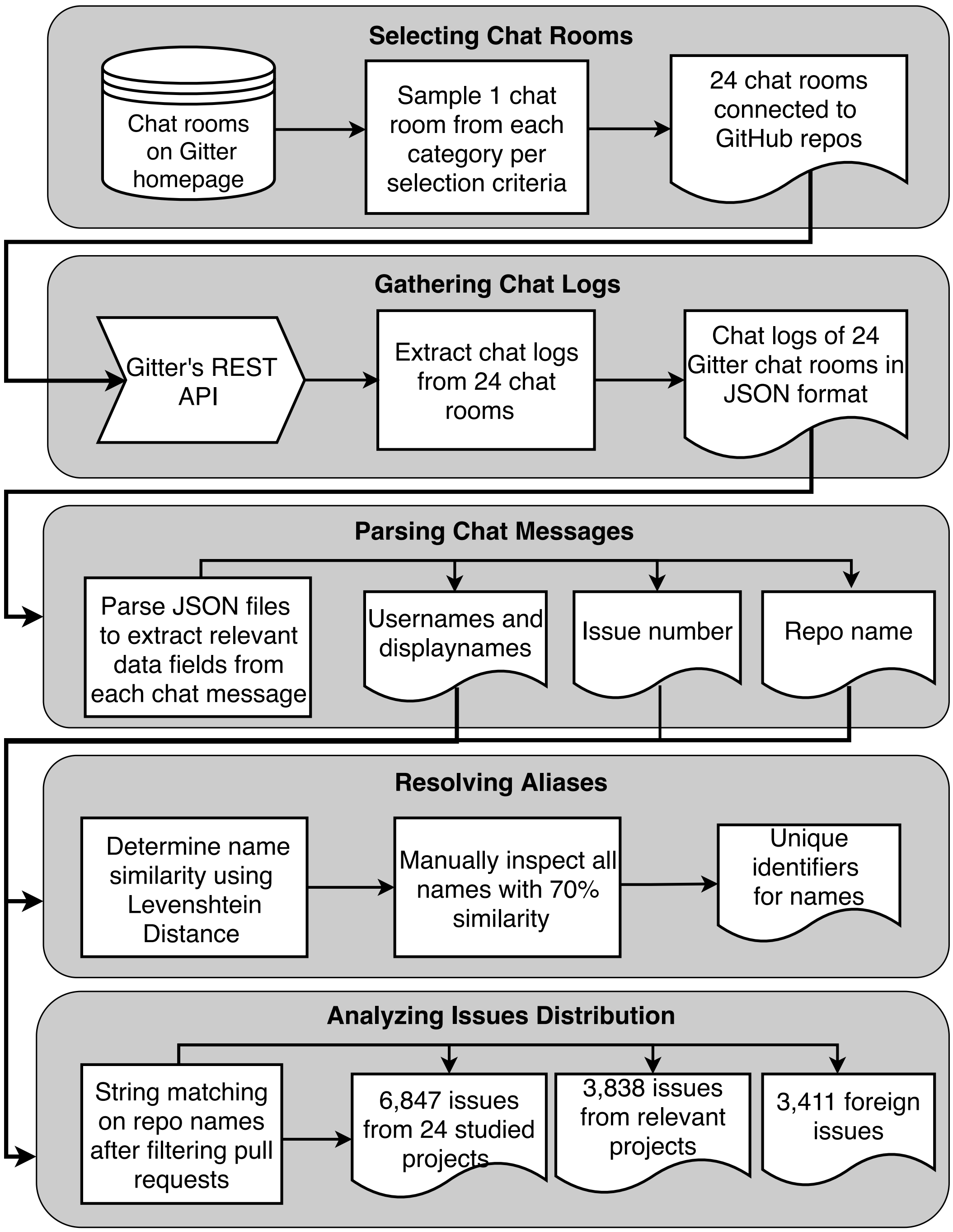


Figure 1: Overview of Methodology

1. that are directly linked to a project's GitHub repository;
2. if the linked GitHub repository contains issue reports;
3. if the chat rooms contain mostly English discussions.

Following these guidelines we randomly sampled 24 chat rooms from various programming communities created around the most popular Gitter tags, and that are available on the Gitter explore page.[8] The selected chat rooms have different numbers of participants which allows us to analyze discussions about popular as well as not very popular projects on Gitter. Table 1 shows the selected chat rooms along with their details such as the number of participants and messages. Each room has the same name as its linked project repository e.g., `amber` in the `amberframework` community is linked to the `amberframework/amber` repository on GitHub.

4.2. Gathering Chat Logs

Gitter provides an API that can be leveraged to obtain chat logs from public chat rooms joined by a user. For the purpose of this study, we joined the chat rooms and used the Gitter API to gather chat logs for the entire period of room existence. The obtained data was saved in the JSON format. An example JSON file containing a chat message from one of the studied chat rooms is shown in Listing 1. It can be seen that in addition to the text message, a chat message also contains participant's id and name, the message timestamp, issue report links, and any other referenced URLs.

4.3. Parsing Chat Messages

Gitter allows linking to GitHub issue reports (by typing # followed by the issue report number) in the linked Git repository, with hovercards enabling a preview of the issue report. The "issues" tag in the JSON object holds the issues that are referenced in a chat message. The object may contain one or multiple issue references along with the name of the repository where the issue

[8]https://gitter.im/home/explore/

Table 1: Studied Gitter chat rooms

Chat Room*	# Participants	# Messages
`amberframework/amber`	379	22,645
`angular/angular`	20,341	1,067,711
`appium/appium`	3,499	41,526
`aws/aws-sdk-go`	953	3,363
`deeplearning/deeplearning4j`	7,898	414,214
`dotnet/corefx`	2,001	22,683
`fossasia/open-event-android`	721	9,408
`google/material-design-lite`	4,936	7,590
`gulpjs/gulp`	3,041	8,808
`kriasoft/react-starter-kit`	1,583	3,477
`magento/magento2`	915	4,999
`mailboxer/mailboxer`	108	139
`meteor/meteor`	3,323	46,385
`Microsoft/TypeScript`	7,478	211,222
`MonoGame/MonoGame`	749	46,001
`openzipkin/zipkin`	1,882	100,556
`patchthecode/JTAppleCalendar`	381	34,609
`PerfectlySoft/Perfect`	1,737	6,555
`scala-js/scala-js`	3,337	85,798
`shuup/shuup`	310	3,507
`TheOdinProject/theodinproject`	10,904	416,408
`twbs/bootstrap`	9,391	17,554
`vuejs/vue`	18,345	387,274
`webdriverio/webdriverio`	4,949	170,674
Total Count	109,161	3,133,106

* Also represents the name of the linked GitHub project repository.

```json
{
"editedAt": "2015-06-06T16:58:00.150Z",
"fromUser": {
"displayName": "Szymon Kazmierczak",
"id": "54c65e95db8155e6700f18b3",
"url": "/Simon-Kaz",
"username": "Simon-Kaz",
},
"id": "5573267c463d0c7c066e46cf",
"issues": [
{
"number": "5218",
"repo": "appium/appium"
}
],
"mentions": [ ],
"readBy": 18,
"sent": "2015-06-06T16:57:32.150Z",
"text": "has anyone had this issue? https://github.com/appium/appium/issues/5218",
"unread": false,
"urls": [],
"v": 2
}
```


Listing 1: An example Gitter chat log in JSON format

report was submitted. The Gitter chat logs also contain a *username* and a *displayName*, as shown in Listing 1, for each message that is posted to the chat room. For a chat message referencing an issue report, we obtained the issue number, repository name and both the *username* and the *displayName* of its author by parsing the logs, and stored the collected data on a per-room basis.

4.4. Resolving Aliases

Prior work suggests the existence of aliases on social platforms including developer collaboration forums, because participants on these platforms can assign themselves multiple nicknames. To reduce the bias in analysis due to multiple identities of the same person, we resolved aliases using the Levenshtein distance [29]. The Levenshtein edit distance is a string similarity metric and we used it to determine the similarity between the names of participants in a chat room, a repository and across them. Bird et al. [20] and Panichella et al. [7] also employed the same similarity metric in their studies for alias resolution. Furthermore, we manually inspected all the names that exhibit 70% or higher similarity to decide if two names belong to the same person. We chose a low similarity threshold to detect all the possibly similar pairs of names which came out to be 494. These names were further manually scrutinized by the authors to reach a final list of 67 aliases. During manual inspection:

- We looked for a similarity between full names. Following this rule Paolo G. Giarrusso and Paolo Giarrusso, Steve Sly Williams and Steve Williams, and, MortenGregersen and Morten Bjerg Gregersen are considered aliases.
- We consider names to be aliases if there is similarity between both the first and the last name. Following this rule Gurch Rai with Gurchet Rai are aliases.
- We do not consider names to be aliases if only first name or only last name, is similar e.g. Simon Sheridan and Simon Brewster are not aliases because only first names are similar and the last names are not.
- We consider names to be aliases if both their username and the display-Name parts match e.g. pfrankov, Pavel Frankov aliases frankpf and Frank because both the usernames and the displayNames are similar.

4.5. Analyzing Issues Distribution

We obtained a total of 20,118 references to issue reports and pull requests in the 24 chat rooms by parsing the JSON and used the repository names to identify the project that the issue belongs to. Our investigation revealed that about 3,966 entries listed in issues were actually not issues. These were either falsely captured as issues due to the use of the # symbol in the text or markdown code (see Section 4.3) or the *repo* field in the data remained empty due to an API bug. We discarded such references, which left a total of 16,152 actual issues. The number of actual issue references found in the chat rooms ranges from 3 to 5,323. As shown in Figure 2, the largest number of issue report references were found in `angular/angular` which is a popular chat room, followed by `openzipkin/zipkin` which has 2,488 references to issues. However, an analysis of issue repositories indicates that the referenced issue reports come from a diverse set of repositories. Only 6,847 issue reports, and 2056 pull requests discussed in the chat rooms come from the GitHub repositories directly linked to the chat rooms. In the rest of the paper we only consider the project issue reports. These issue reports are subsequently referred to as *Gitter-issues* and do not contain any references to pull requests.

Issues are also referenced in the chat rooms across repositories (see Figure 3), and we refer to such issues as *foreign* issues. In many cases, a referenced issue report belongs to a repository that has a relation with the chat room-connected repository, e.g., both repositories host sub-projects of a main project or both repositories have the same parent repository. For example, the `meteor` project has several repositories (e.g., `meteor/meteor-feature-requests`) that contain artifacts for the main project (`meteor/meteor`). We do not consider issue reports that are referenced across such repositories as foreign. To detect foreign issue reports, we did a simple substring comparison to determine whether the chat room-connected repository and the repository of the referenced issue report share a common parent-level project or repository. For example, the aforementioned example would not be considered a foreign issue report since the repositories share the common `meteor` prefix, while the issue report `yeoman/generator-webapp/issues/342` referenced in the `vuejs/vue` chat room would be considered foreign to the `vue` project. We found a total of 3,411 foreign issues, which is 21% of the total number of referenced issues in the studied chat rooms.

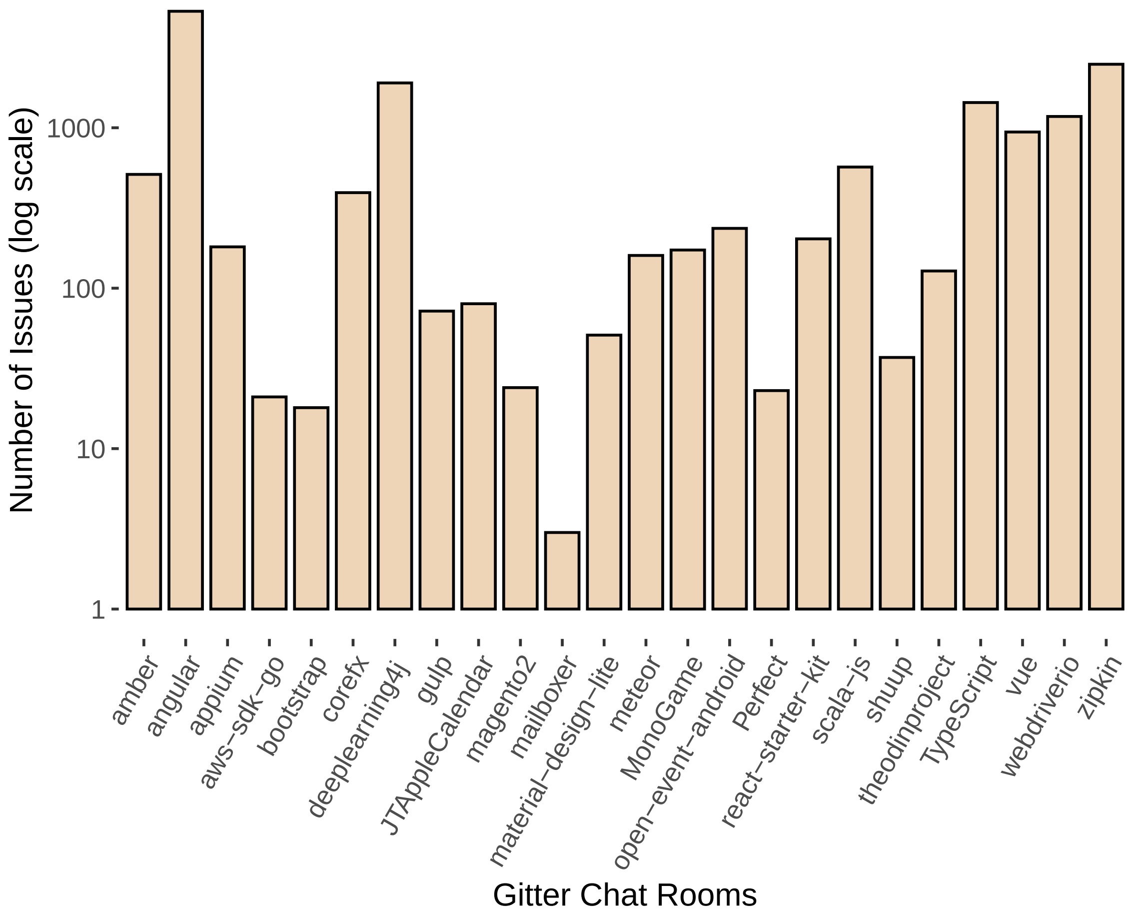


Figure 2: Total number of issue references found in the 24 studied Gitter chat rooms using log scale.

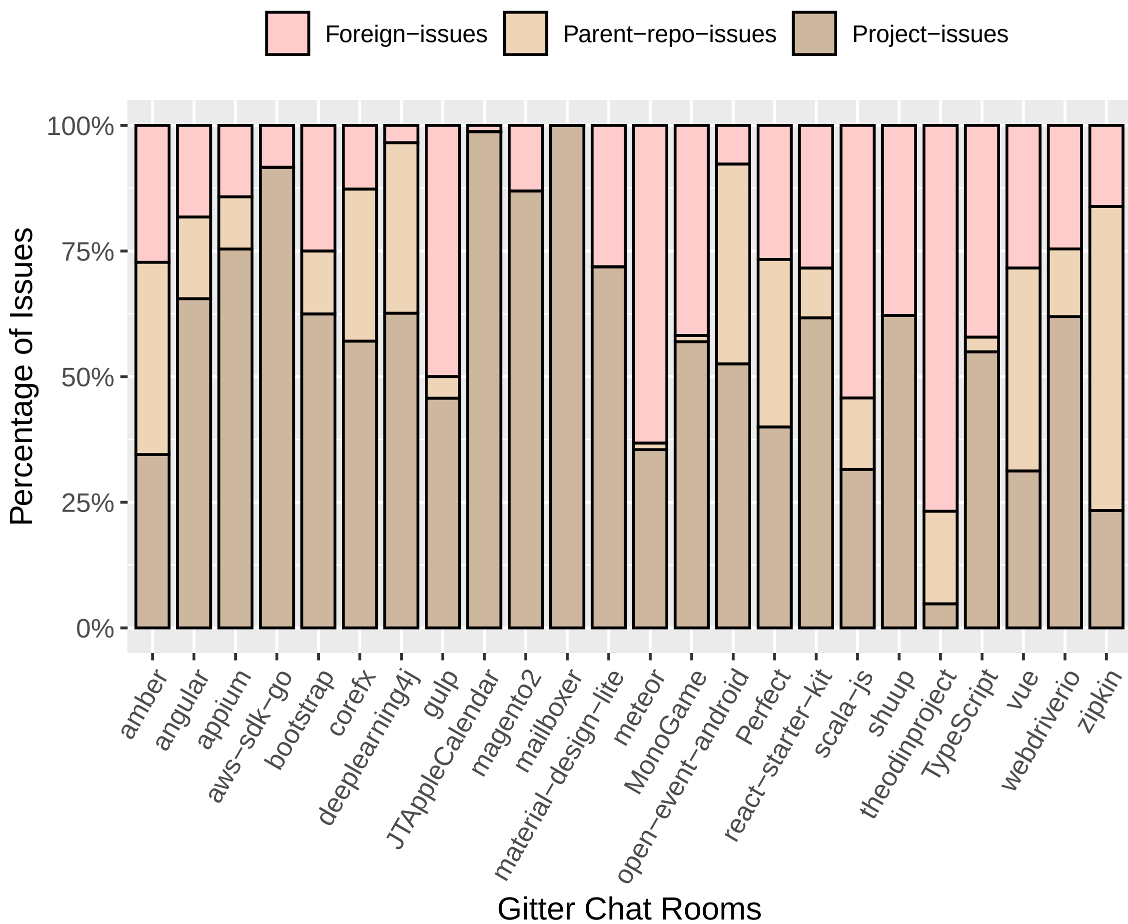


Figure 3: Distribution of the percentage of issues in 24 studied Gitter chat rooms

5. Study Findings

5.1. RQ1: Who refers to and discusses issue reports in Gitter chat rooms?

Motivation: Coordination and communication among project contributors is crucial for the success of a software project. Prior work has established that open-source developers employ different mechanisms for communication. To improve our understanding of the extent to which Gitter is used for critical project communications, we explore issue discussions and their participants. Identifying people who mainly contribute to Gitter issue discussions offers deeper insights and allows us to reason about the usefulness of newly introduced Gitter in open-source software development.

Approach: Each studied Gitter chat room hosts chats on one specific open-source project, which uses GitHub as its version control system. GitHub defines a repository *collaborator* to be someone on the core development team of the project who has commit access to the main repository of the project. A *contributor* is someone from outside the core development team of the project who contributes changes to the project. Following GitHub's definition and the more elaborate structure proposed by previous work [30], we refer to collaborators as the small group of core-developers who have direct access to the source code repository and control the project. Then there is another group of external developers who file issue reports or make minor fixes. These are called contributors and the changes made by them are reviewed by the core developers before they are accepted to become a part of the project. Finally, there is a group of end-users, who do not actively participate in the development, but use the software, file issue reports, and are part of the community. We also refer to them as end-users in the paper.

We obtained the names of project collaborators and contributors from the project's GitHub repository by leveraging the GitHub API. While obtaining the list of collaborators we considered all those who have commit access to the repository whereas for the contributors, we considered all the names listed under the contributors page in the official GitHub repository of each project.

At the same time, for each issue referenced in a Gitter chat room, the *username* and the *displayName* of the person referencing the issue was also saved. Note that we do not consider everyone involved in the discussion but only the person who wrote the message containing reference to an issue report. We then used a Python program to discover exact matches between the names of participants who referenced issue reports in chat rooms and the project contributors. Finally, we calculated the percentage of referenced issues by the project collaborators, contributors and end-users in the 24 Gitter chat rooms.

Results: **End users referenced the majority of the issue reports in the studied chat rooms as shown in Figure 4**. Overall, more than 50% issue reports were referenced by the end-users in 14 out of 24 chat rooms. `theodinproject` chat room is the only one where all issues were referenced by the project collaborators and contributors. In `zipkin` and `corefx`, collaborators and contributors referenced around 75% issues. The remaining issues were referenced by participants who do not directly contribute to the project or it could be the case that these participants were not identified due to aliases.

In particular, `appium` has 99% issues referenced by people who are not contributors or collaborators but instead end-users. Similarly, `gulp` and `vue` have a very small number of issue reports referenced by the project collaborators and contributors, i.e., between 5% to 12%. We anticipate that the percentage issues referenced by contributors will increase if we consider all participants involved in issue report discussions instead of just considering participants who directly reference issue reports in their messages. Regardless of that, the small number of issue discussions by the developers of these 3 projects could be attributed to the limited reliance of these projects on Gitter as a communication channel. Our assumption was confirmed when we found that `vue` and `appium` have their official channels on Discord[9], and Discuss[10] respectively and these projects do not officially use Gitter for collaboration. The presence of multiple chat com-

[9]https://discord.com/invite/HBherRA
[10]https://discuss.appium.io/

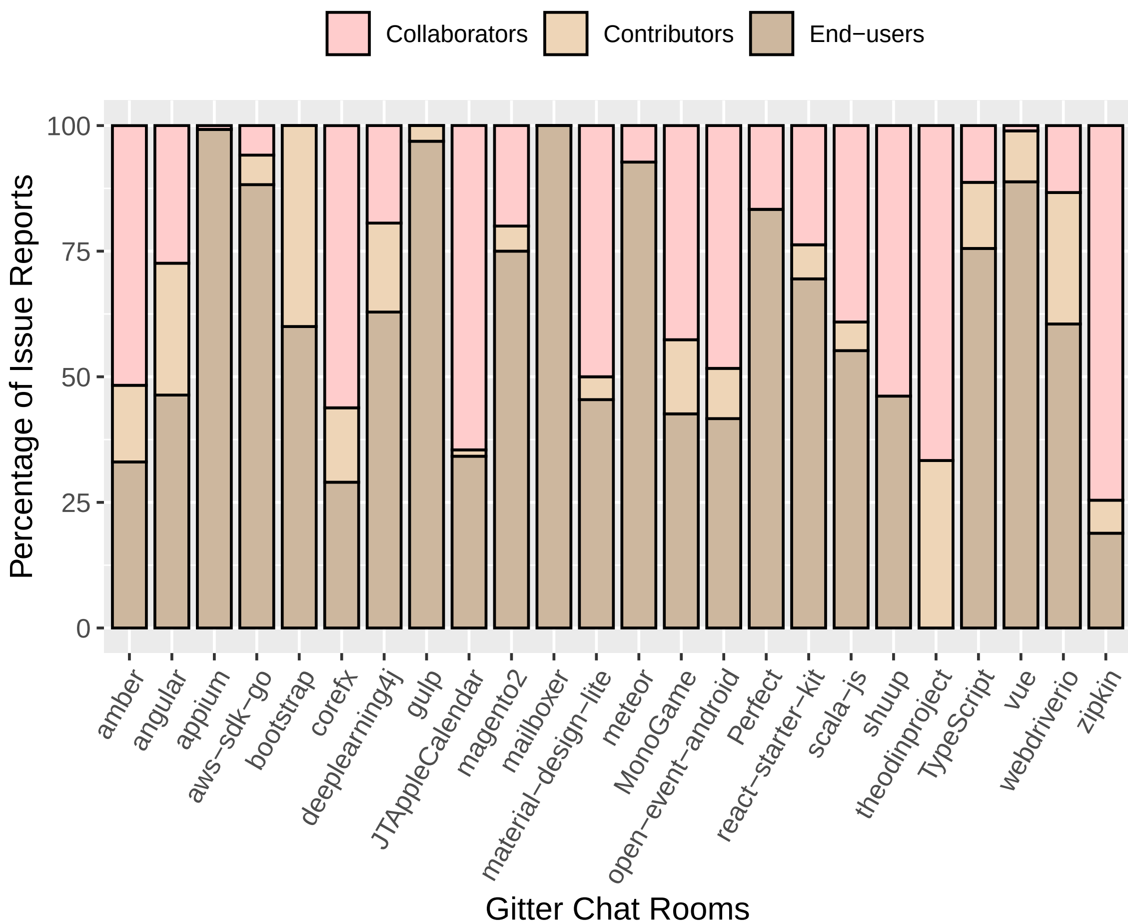


Figure 4: Distribution of the percentage of issue reports referenced in the 24 Gitter chat rooms by the actual project collaborators, contributors and end-users.

munities of these projects suggests that developers might also be talking about important project decisions at other places, and hence we observed limited issue report discussions or activity in the Gitter chat rooms of these projects. Therefore, we conclude that Gitter is actively used by open source project developers to collaborate on project activities if it is adopted as the main collaboration tool for the project whereas the presence of multiple channels across platforms can limit the developer activity on Gitter. Future studies should investigate in detail how having multiple chat channels affect the communication and collaboration activities on Gitter.

Lastly, we observed the presence of bots in 4 chat rooms while analyzing the names of Gitter users. The bot referenced 108 issue reports in `amber`, 46 in `deeplearning4j`, and 80 in `JTAppleCalendar`. The *odin-bot* did not reference any issue reports in `theodinproject` chat room.

5.2. RQ2: What is discussed about issue reports on Gitter?

Motivation: The topics of developer discussions on a platform are a quick indicator of its usefulness in communication and collaboration activities. Due to the importance of issues, discussions around them can reveal interesting insights about a project e.g., information about how bugs are resolved in an OSS project can be found in these discussions.

Approach: To understand the purpose of issue references and discussions we carried out a qualitative study of a statistically representative sample of issue reports in the 24 Gitter chat rooms. From the entire data set of *Gitter-issues*, we randomly selected a representative sample of 364 issues, with 95% confidence level and a 5% confidence interval for manual analysis. The selection of a statistically significant sample size based on population size, confidence interval, and confidence level was introduced by Krejcie and Morgan in 1970 [31] and since then it has been employed by many studies in the past [32, 33, 34, 35]. Our sample contains an equal number of issues i.e., 16 from each room except for the projects, such as `mailboxer`, which do not have enough issue references. In that case, we studied all the discussed issues. After selecting our sample of

Table 2: Categorization of the purpose of referencing issues in Gitter and the distribution of the percentage of identified categories.

Category	Description	% Issue Reports
Technical support	Reference to an issue that contains additional information related to problem	37%
GitHub issue tracker activity	References due to opening, closing, or commenting on issue reports	23%
Inquiries about issue state	Discussion about when issue will be solved or integrated into the main project	14%
Contributions, suggestions or feedback	Asking for contribution or feedback on the referenced issues	12%
Project updates and future plans	Issue references to provide updates or to discuss future plans e.g. release schedules	7%
Not clear	Purpose of referencing an issue report could not be inferred	6%
Request supplementary bug fixes	References to issues that are resolved but still contain bugs	1%

issue references, we manually labelled them with the goal of understanding the purpose of references in the chat room messages or discussions.

Starting from the chat message that directly references the issue report, we read the entire discussion around the issue report. This involved reading neighbouring messages that were posted in the chat room before and after the message containing an issue reference. Since discussions around referenced issues were interleaved with other chat messages, it was sometimes difficult to identify the context. In such cases, we leveraged the name of the participants and the time stamp of the messages, and read the discussions until the participants involved in the issue report discussion no longer appear in the chat or the time stamp indicated that the discussion was likely to have been initiated at that point.

The process of labelling was iterative, so initially 2 authors independently labelled 13 issues followed by another 50 issues. They started with an unknown category list and finalized the categories in 2 iterations, following an approach similar to open coding [36]. While reading the discussions, the authors asked the following question: "What is the purpose of referencing the given issue in this chat message?". They identified the purpose and the context of referencing an issue report in the discussions, and categorized it accordingly. In cases where the identified category did not match an existing one, we added a new category to the list and restarted the labelling process with the new category list. The final list of categories was finalized by 2 authors of this paper who both independently labelled issue report discussions with a Cohen's Kappa [37] inter-rater agreement of 0.8. A substantial agreement was achieved because the purpose of referencing an issue report is often explicitly mentioned in the chat by the participant who references the issue report, or it could be identified from the discussion. The small number of conflicts that appeared in the categorization were due to a new category that was identified by one of the authors and not by the other. These conflicts were resolved by the authors by including the additional category, and a final categorization scheme consisting of 8 categories was agreed upon (see Table 2) before the remaining issue references were labelled by one of the authors.

Further, to get insights about the reasons of referencing foreign issues in chats, we also manually labelled a statistically representative sample of 93 foreign issues with a 95% confidence level and 10% confidence interval.

Results: The final list of categories that emerged from our manual labelling is shown in Table 2. The most common purpose of referencing issues in chat messages is to acquire ***technical support***. In regards to issues, technical support is frequently (37%) sought in chat rooms, often by product users, to resolve source code errors, or to develop better understanding of the syntax and semantics of a programming language. For example, while resolving an error, a participant wrote in chat: *"I'm having some issue with adding a <tr> to the table using js ... the checkbox wouldn't show up ... but i couldn't find any fix for it. please refer to this link[11], if you think you can help."*

As shown in this example, issue reports are referenced in the questions to find a workaround for the issue if it is yet to be fixed or when the solution proposed in issue report comments does not work. Similarly, while answering questions, issue reports are referenced to notify participants of the progress made so far on an issue or to communicate that the issue report belongs to the WONTFIX category. Sometimes, however, participants seeking help mention problems relevant to the project which the developers have not come across previously. Such questions lead to the creation of new issue reports in the issue tracker and we assign them 2 labels: *technical support* and *GitHub issue tracker activity* as such references have a dual purpose.

GitHub issue tracker activity is the next popular category and discussions around it constitute for around 23% of the issue references. Activity refers to anything happening to an issue report in the GitHub issue tracker and the largest number of references in this category were due to opening, closing and commenting on issues whereas discussions related to issue trolling also appeared but rather less frequently. The following is an example of a chat message that

[11] `https://github.com/angular/angular/issues/5917`

is relevant to this category: *"Yep, so that guy's comment is valid: link."*[12]

Issues are often opened due to incorrect outputs, crashes, memory leaks, unsuccessful builds, or to request a feature enhancement or a document update. Sometimes people also mistakenly open issues which they consider to be bugs or in plea for help. We observed during our manual annotations that such issues are quickly identified and closed by the developers, hence preventing huge backlogs and improving the overall bug triaging process.

Almost 14% of the issue reports referenced in the chats are actually updates about work in progress or ***inquiries about issue state*** such as the following *"https://github.com/webdriverio/webdriverio/issues/996 seems to indicate there isn't a way to do this, but since it has been there for over an year, I was wondering if that is still the case".* These questions often directly address the project contributors and are followed by explanations of what might be delaying the resolution of an issue or when is it likely to get resolved and whether it is still relevant or not. A closely-related category involves referencing issue reports to communicate **project updates and future plans** including inquiries related to upcoming release schedules and changes from prior release. We group these 7% of issues separately as they are related to the future changes planned for the project and have nothing to do with the current activity in a project or a specific issue report. Nonetheless, both of these categories together serve to improve the awareness of project stakeholders about the current state of issues and provides them a way to keep track of progress.

Requests for supplementary bug fixes account for 1% of the issue report references. Prior work suggests that 22% to 33% of resolved bugs involved more than 1 fix attempt [38]. For example, this could be due to the missed porting changes, incorrect handling of conditional statements, or incomplete refactoring. Moreover, a bug fix could be incomplete and even introduce new bugs. Our manual analysis confirms that developers close issues that still contain errors or omissions or that lead to the introduction of more bugs. Consequently bug fixes

[12]`https://github.com/reponame/issues/1392#issuecomment-450722812`

are required even after an issue has been marked as resolved or closed. Issue reports referenced in discussions during the bug fixing process or requests for re-verification of the closed issues were categorized as supplementary bug fixes.

Project developers also ***request contributions, suggestions and feedback*** for a particular issue in the chat rooms. Around 12% of the total issue reports were referenced in chats to ask for contributions and opinions. During the manual annotations we observed that suggestions and feedback is usually provided by the development team members which is not surprising. However, due to the public nature of the Gitter chat rooms, participants other than the development team members offer contributing to an issue. Moreover, we also noticed that the project collaborators encourage contribution from participants external to the team (i.e. those with no contribution to the project), thus lowering the entry barrier for new contributors.

Sometimes a referenced issue is a link only without associated discussion or the purpose of reference cannot be identified from the surrounding context. We categorize such references as ***not clear.***

The ***foreign issues*** come from a multitude of repositories and are referenced in the chats for all sorts of reasons. The top reasons for referencing foreign issues are to:

1. Discuss a project's external dependencies and updates affecting a project;
2. Provide a minimal reproduction using a sample repository on GitHub;
3. Share information and experiences gained from other projects;
4. Propose a feature similar to another project or figure out how someone else implemented a similar feature;
5. Discuss Gitter, GitHub or Travis-CI related issues affecting a project;
6. Ask for help with installations and project configurations;
7. Informal discussions around an issue or random commenting.

5.3. RQ3: How does the issue resolution time of issues that are discussed and not discussed on Gitter differ?

Motivation: In this research question, we examine if issue report discussions on Gitter are correlated with a faster issue resolution process. Through this question we investigate how the adoption of Gitter affects part of the software development life cycle associated with maintenance and bug fixing.

Approach: To find out if issue report discussions in Gitter reduce the time to resolve an issue, we compared N *Gitter-issues* with N randomly selected issues referred subsequently as *Random-issues*. The random issues were extracted from the GitHub repository linked to the chat room and excludes issues referenced in Gitter chats. Our sampling methodology ensures that equal number of issues are included from each repository in the *Gitter-issues* and *Random-issues* data set. However, the number of issues included from each repository varies, and depends on the actual issue references to that repository in its chat room.

Next, we obtained the metadata of all the *Gitter-issues* and *Random-issues* from their respective GitHub repositories using the GitHub API. The collected data was parsed and the issue resolution time of all issue reports was calculated using the *created_at* and *closed_at* dates. The resolution time of issue reports is the time period between the submission of an issue on GitHub to the time when the issue was resolved and the fix was accepted. Our null hypotheses is:

H_0 : *There is no difference between the issue resolution time of Gitter-issues and Random-issues.*

H_a : *Gitter-issues have a longer issue resolution time than Random-issues.*

To test our null hypotheses, we used the Mann-Whitney test which is a non-parametric alternative to the t-test and is used for non-paired data. We chose this test because the issue resolution time in our data set does not follow a Gaussian distribution according to the results of the Kolmogorov-Smirnov test for normality [39]. Furthermore, we employed multiple independent Mann-Whitney tests [40] followed by Bonferroni correction to test our hypothesis while controlling for the four important confounding factors. We chose an $\alpha = 0.05$ which becomes $\alpha = 0.003$ (i.e., 0.05/16) after the Bonferroni correction. To measure

Table 3: Description of the four confounding factors included in study. Each confounding factor has two possible levels as shown in the third column

Confounding Factors	Type	Levels	Description
Comments	Categorical	$>$ mean number of Gitter/Random issue comments $\leq$ mean number of Gitter/Random issue comments	Total number of comments on an issue in GitHub issue tracker
Reporter	Categorical	None Not None	The person who reported the issue is associated with the project (Collaborator, Contributor, Member, Owner) or not (None)
Milestone	Categorical	Yes, No	Issue's milestone existed or not
Assignee	Categorical	Yes, No	Whether someone was assigned the responsibility to move the issue forward or not

the effect sizes of the differences between *Gitter-issues* and *Random-issues* we used *Cliff's delta* along with interpretations of Romano et al. [41] based on which the difference is considered to be Negligible if Cliff's $|d| \leq 0.147$, Small if Cliff's $|d| \leq 0.33$, Medium when Cliff's $|d| \leq 0.474$, and Large otherwise.

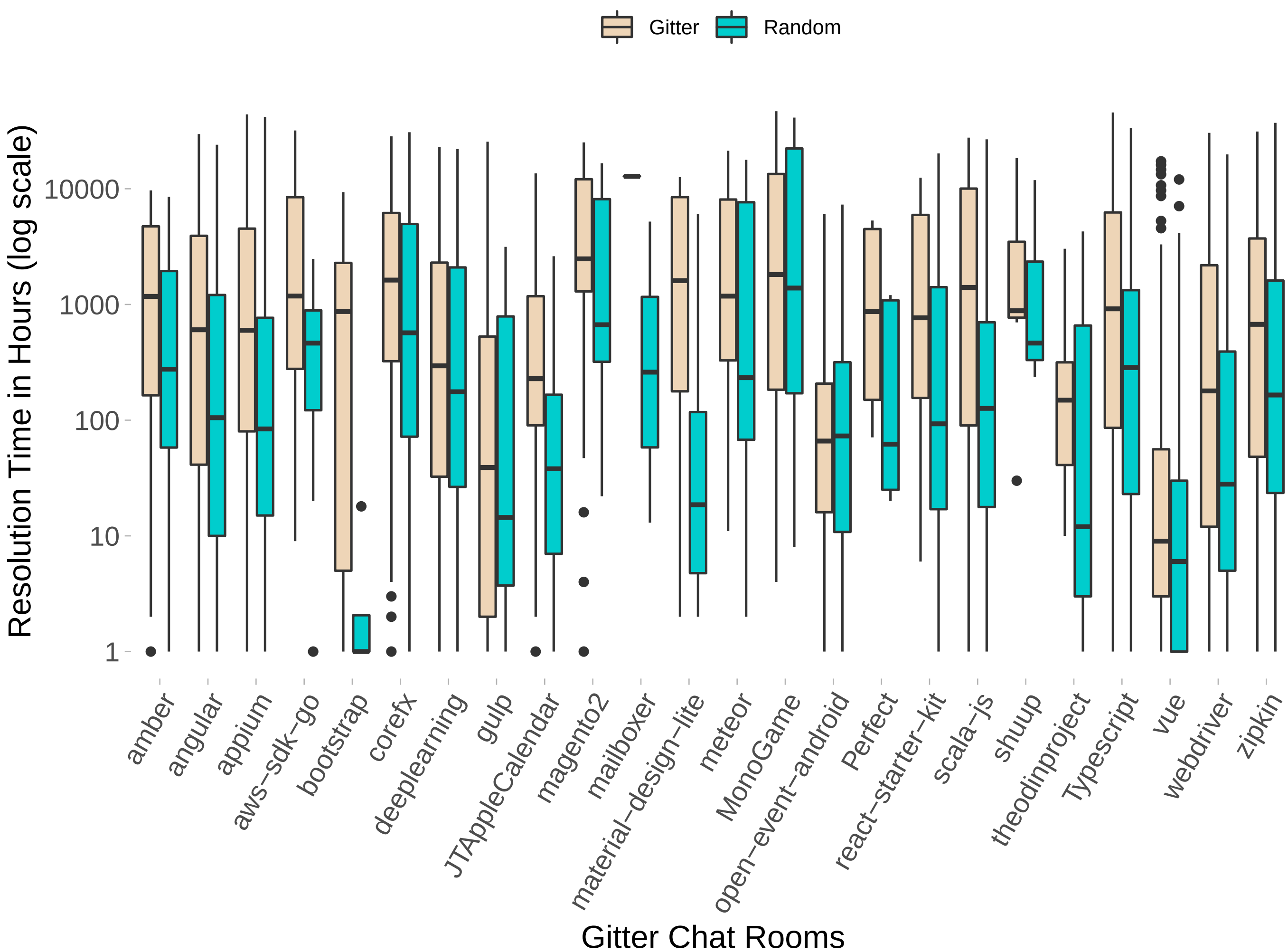


Figure 5: Comparison of resolution time of issue reports referenced in Gitter chat rooms and the issue reports randomly sampled from linked GitHub repositories. The X-axis shows the name of Gitter chat rooms. The Y-axis represents resolution time in hours shown in log scale.

The four confounding factors that we controlled for in our study include *issue comments*, *issue reporter*, *issue milestone* and *issue assignee*, as shown in Table 3. We considered these four factors in our study based on the findings of previous studies where number of issue comments [42], [43], [44] and the issue reporter's reputation [42] were found to have the most impact on the time to fix an issue. Similarly, issue assignee and milestone were also suggested as top predictors of issue fix time by prior studies [44, 45].

Results: Figure 5 shows a comparison of the resolution time of *Gitter-issues* and *Random-issues* on a per room basis. Table 4 presents the summary statistics of the issue report resolution time and the number of open issues in the analyzed data set. Among the analyzed data, 16.1% issue reports in *Gitter-issues* and 12.9% in *Random-issues* are still open. To our surprise, the resolution time of our random sample of issue reports is smaller, with a mean issue report resolution time of 1,526.3 hours, compared to the issue reports referenced in the Gitter chat rooms, which have a mean of 2,909.9 hours. The medians also show a similar relationship.

Table 5 presents the results of multiple Mann-Whitney tests that we carried out to check the statistical significance of differences between the resolution time of *Gitter-issues* and *Random-issues*. The null hypothesis was rejected according to the results of the Mann-Whitney tests $(p - value < 0.003)$ for 11 out of 16 issue report groups confirming that the observed differences are statistically significant at an $\alpha = 0.003$. Furthermore, the *Cliff's delta* results suggest that the difference in the resolution time mostly has *Small* effect sizes within groups.

To avoid bias in comparison, we calculated the difference of median resolution time of all *Gitter-issues* and *Random-issues* by bootstrapping the latter 1,000 times with a 95% confidence interval and a 5% confidence level lies in [174.5 251.5]. We used median instead of mean because of the high skewness of our resolution time data. The reported difference suggests that Gitter-issues have a higher median resolution time and as the confidence interval does not include 0, we conclude that the difference is significant. There are two possible explanations to this, one is that difficult issues are more frequently discussed on

Table 4: Summary of resolution time of issue reports in *Gitter-issues* and *Random-issues* in hours

Statistic	Gitter-issues	Random-issues
Minimum	0.0	0.0
1st Quartile	21.0	5.0
Median	363.5	72.0
Mean	2,909.9	1,526.3
3rd Quartile	3,580.2	863.5
Maximum	46,797.0	41,786.0
Open issues	16.1%	12.9%

Gitter, and therefore their resolution time is also longer. This also seems to be a reason for the comparatively larger number of non-closed issues in our sample of *Gitter-issues* in relation to the *Random-issues*. Another reason contributing to the delayed resolution of *Gitter-issues* could be the lack of attention received by these issues, which motivates our RQ4.

5.4. RQ4: Does the discussion on Gitter impact the issue's activity and resolution time of the issue?

Motivation: We believe that one reason that contributes to the delayed resolution of *Gitter-issues* could be the lack of attention received by these issues. This motivated us to analyze the issue comment activity in the GitHub issue tracker.

Approach: The GitHub issue tracker provides support for adding comments to the issues, and stakeholders use this feature to discuss issues. Issue comments may contain useful information [46] and developers can make use of this information such as when discussing design ideas and implementation details to resolve issues in a timely manner. In this research question we analyzed the distribution of comments in the GitHub issue tracker of issues that were never referenced on Gitter. We extracted the comments from each *Gitter-issue* using

Table 5: Summary of Mann-Whitney U-test for comparison of the resolution time of *Gitter-issues* and *Random-issues* while controlling for four confounding factors: issue comments, issue reporter, issue assignee and issue milestone. The bold values indicate statistically significant differences at α=0.003.

Comments	Milestone	Reporter	Assignee	Mann-Whitney
$\leq$ mean	No	None	No	**2.2E-16**
$\leq$ mean	No	None	Yes	7.87E-01
$\leq$ mean	No	Not None	No	**2.01E-03**
$\leq$ mean	No	Not None	Yes	5.90E-02
$>$ mean	No	None	No	**2.20E-16**
$>$ mean	No	None	Yes	**1.71E-06**
$>$ mean	No	Not None	No	**1.38E-03**
$>$ mean	No	Not None	Yes	**2.43E-06**
$\leq$ mean	Yes	None	No	**1.59E-05**
$\leq$ mean	Yes	None	Yes	3.27E-01
$\leq$ mean	Yes	Not None	No	**6.24E-04**
$\leq$ mean	Yes	Not None	Yes	**5.83E-07**
$>$ mean	Yes	None	No	**4.27E-06**
$>$ mean	Yes	None	Yes	1.60E-03
$>$ mean	Yes	Not None	No	**7.37E-06**
$>$ mean	Yes	Not None	Yes	3.31E-03

the GitHub API. We then compared the date of issue reference on Gitter with the comment creation date to obtain the value of time difference between an issue reference in a Gitter chat room and an issue comment in the GitHub issue tracker. We calculated the number of issue comments that were made within one week of an issue's reference on Gitter. Finally, we calculated the Comments Change Ratio as follows:

$$\textit{Comments Change Ratio} = \frac{\text{\# of comments after Gitter reference}}{\text{\# of comments before Gitter reference}}$$

The Comments Change Ratio shows how the number of comments on an issue changes within one week after and before an issue's reference on Gitter. A ratio larger than 1 indicates that there were more comments on an issue in the week succeeding its Gitter mention as compared to the preceding week. A ratio smaller than 1 indicates the opposite whereas a ratio equal to 1 means the number of comments before and after were equal.

Results: GitHub issue tracker's comment feature is used by all 24 studied projects to discuss issue reports. The median number of comments on the *Gitter-issues* ranges from 1 to 16. Table 6 shows how the mean number of comments changes in the one week time period before and after an issue is referenced on Gitter. For a vast majority of *Gitter-issues* that had at least some activity on GitHub, we found more comments in their issue tracker after the issue report was referenced on Gitter. This led to a higher mean number of issue comments in the third column in Table 6 as compared to the second column, for some of the projects.

Figure 6 further shows the distribution of the Comments Change Ratio of *Gitter-issues* on a per room basis. The median Comments Change Ratio is around or above 1 in most cases indicating that issues received more comments in the week following their reference on Gitter than the preceding week. As shown in the last two columns of Table 6, there was a high percentage of Gitter-issues that had no comment activity in the tracker before the issue was brought to Gitter. It appears from the data that issue comment activity increased in

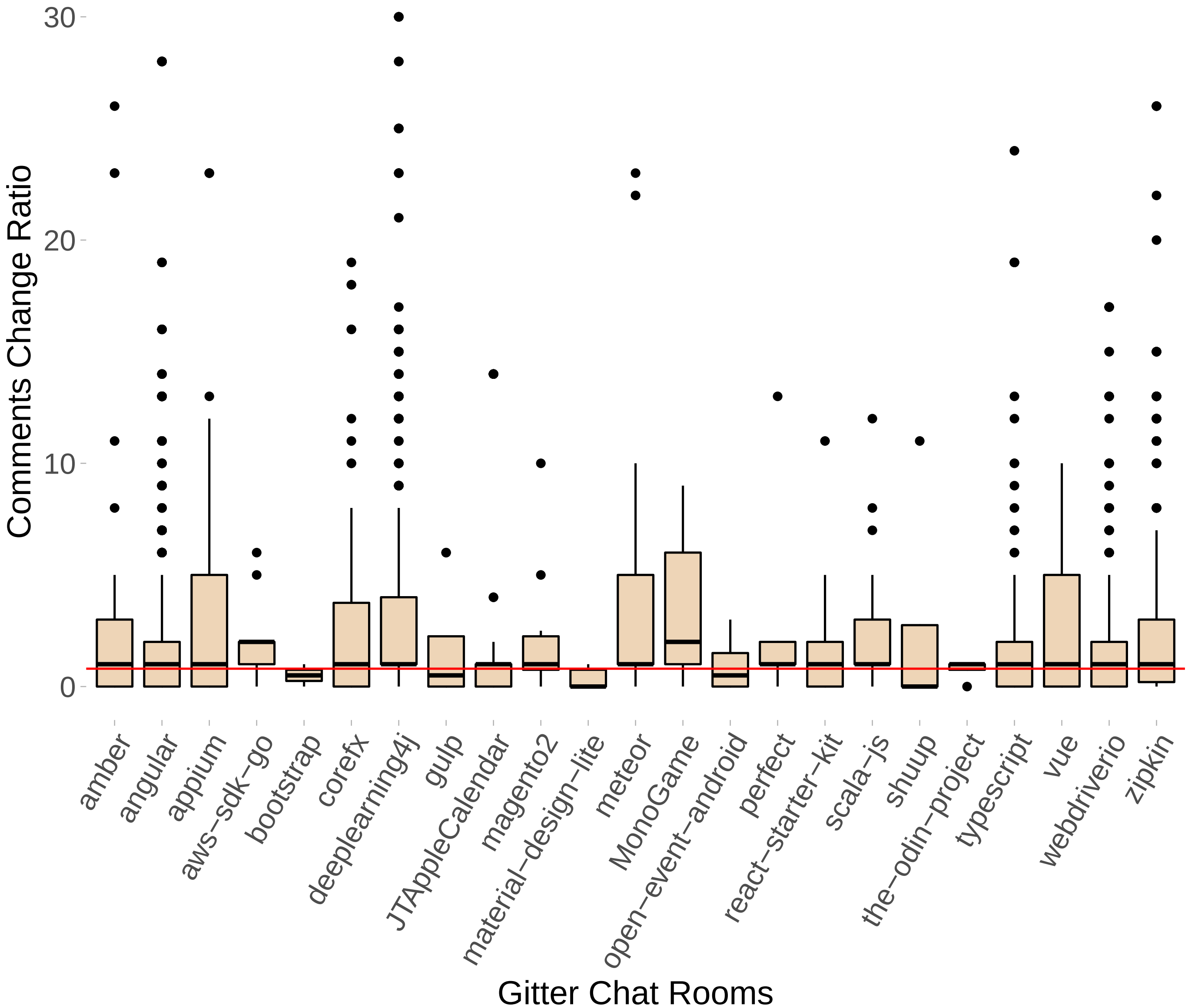


Figure 6: Comments Change Ratio showing ratio of number of issue comments in the GitHub issue tracker one week after and before issue reference in a Gitter chat room. The X-axis shows the name of Gitter chat rooms. The Y-axis represents comments change ratio using log scale. The horizontal red line shows ratio = 1 meaning equal number of comments before and after Gitter reference.

Table 6: Distribution of mean issue comments and percentage issues that have zero comments before and after an issue reference on *Gitter*

	Mean Issue Comments		**% Issues with no comments**	
Chat Room	Before	After	Before	After
amber	1.4	1.4	67.4	57.6
angular	1.5	1.5	64.4	62.3
appium	1.9	1.7	77.6	68.3
aws-sdk-go	0.2	1.2	84.2	52.6
bootstrap	0.1	0.1	88.9	88.9
corefx	0.6	1.4	81.4	66.3
deeplearning4j	1.0	2.1	79.3	57.5
gulp	0.6	0.4	88.9	88.9
JTAppleCalendar	1.1	1.4	57.4	59.2
magento2	0.7	1.6	75.0	55.0
material-design-lite	0.5	0.1	83.3	91.7
meteor	0.6	2.2	84.7	61.0
MonoGame	0.0	1.4	96.8	65.1
open-event-android	0.2	0.2	94.3	91.4
perfect	0.5	3.0	66.7	33.3
react-starter-kit	0.3	0.8	77.6	65.5
scala-js	0.3	0.8	92.3	71.0
shuup	0.4	1.0	72.7	90.9
the-odin-project	0.2	0.8	75.0	25.0
TypeScript	0.6	0.6	81.1	78.5
vue	4.6	5.3	35.9	45.1
webdriverio	0.9	0.9	72.0	69.6
zipkin	1.1	2.0	79.8	53.3

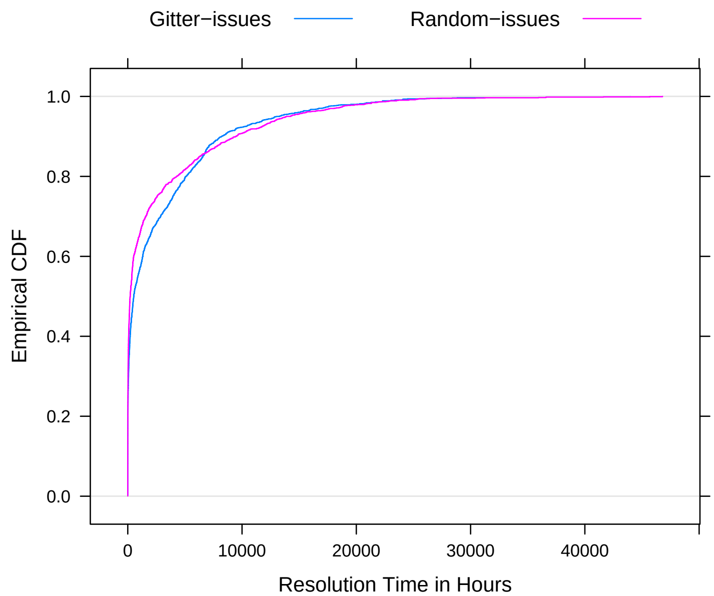
the following week and the percentage of issues with no comments declined. The increase in issue activity could be an indirect consequence of the issue reference on Gitter showing that **Gitter references could possibly trigger activity in the issue trackers.** Additionally, an issue report is referenced on Gitter after a median of 120 hours after its creation but there are some issue reports that are mentioned after a median of 15,024 hours as shown in Table 7. Perhaps these issue reports had been unreasonably prolonged, and were rather referenced in the chats to prevent further impediment and facilitate resolution. The cumulative frequency distribution curves in Figure 7a show that a larger number of *Random-issues* are resolved at any given point in time as compared to the *Gitter-issues* until around 10,000 hours. However, as the time in hours increases beyond that point, the *Gitter-issues* take the lead. The upward trend in the number of issue comments after its reference on Gitter, and the short duration (4,631) between the issue reference and resolution suggest that Gitter reference could be indirectly affecting the long-standing issue resolution, leading to a higher number of resolved *Gitter-issues*. The partial cumulative frequency distribution curve in Figure 7b was drawn to highlight this difference. Based on these observations it appears that **Gitter might be used to revive and facilitate resolution of issues that have not been addressed in a long time**. Since Gitter references serve as a way of bringing the issue reports to the attention of developers, we believe that for long-standing issue reports, the chances of resolution are higher when the issue report is brought to Gitter, which ultimately leads to their resolution. Future studies should further investigate how referencing an issue report on Gitter impacts their activity in the GitHub issue tracker, and whether an earlier reference in Gitter can lead to faster resolution.

6. Discussion and Implications

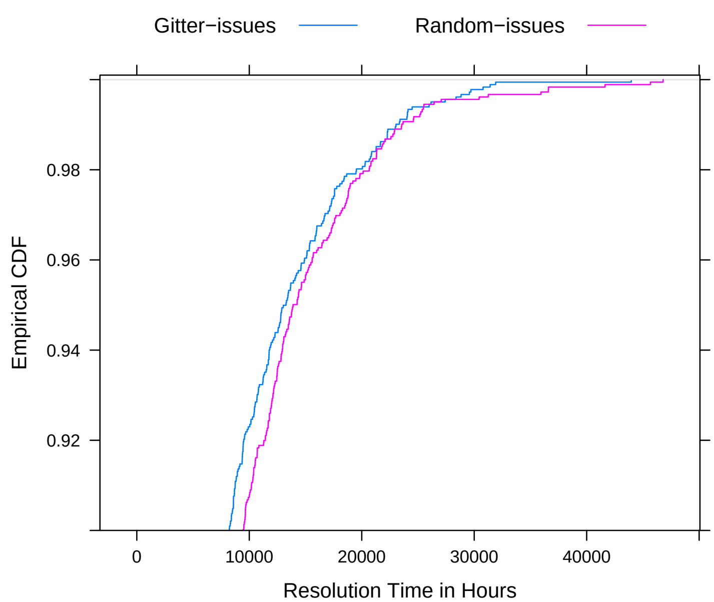
In this section we discuss our findings and their implications for researchers.

6.1. Prevalence of issue report discussions on Gitter

Our analysis revealed that only 0.49% of the messages in the 24 studied chat rooms contain an issue report reference but discussions around these issues



(a)



(b)

Figure 7: Cumulative frequency distribution curve showing the age of Gitter-issues and Random-issues. The X-axis shows the resolution-time of issues in hours and the Y-axis shows the percentage of resolved issues after n hours of creation. Figure (b) shows a zoomed in version of the part (a) after 10000 hours.

Table 7: Summary statistics of the duration in hours between issue creation and reference(ref) and issue reference and closing for the long-standing *Gitter-issues*. Here long-standing issues refer to issues that were not resolved even after 10000 hours of their creation

	Long-standing issues		**Remaining Gitter-issues**	
Statistic	**creation-ref**	**ref-closing**	**creation-ref**	**ref-closing**
Minimum	10,080	0	0	0
1st Quartile	12,192	1,320	0	96
Median	15,024	4,872	120	696
Mean	16,250	6,077	955	2,462
3rd Quartile	18,360	8,448	1,008	3,264
Maximum	41,136	29,640	9,960	31,272

could be significant for the success of a project. The discussions often involve a project's contributors, and contain valuable knowledge about an open source software project. The informal and unstructured nature of Gitter chats allow developers to reveal their opinions about various aspects of a software system at different times and in different contexts leading to recurrent referencing of issue reports in chats.

When developers and end-users discuss current problems in a software system, this occasionally leads to the opening of a new issue. Similarly they discuss existing issue reports to analyze if these issues significantly impact the project and can be prioritized. Discussions around the issue resolution strategies and issue assignments to different contributors also surface in the Gitter chats. One can also find justifications of WONTFIX issues [47][48] and the work around for those.

To obtain statistical evidence of the influence of different distribution of collaborators and contributors in the chat room on the issue resolution time, we used the Spearman correlation test [49]. It is a non-parametric measure of association between two variables. For the purpose of measuring the correlation, we arbitrarily considered 5 messages before and 5 after an issue reference to be part of issue discussion. The number of messages contributed by the developers in the issue discussions were then correlated with the issue's resolution time. Table 8 shows that the Spearman correlation test reveals a statistically significant correlation in only 5 out of 24 chat rooms. Therefore, despite their prevalence in Gitter chat rooms, the issue discussions do not impact issue resolution time in a statistically significant way for most projects.

6.2. ***Gitter as a complementary communication channel***

The top referenced repository in issue report discussions is the GitHub repository linked to the chat room, however, a substantial number of issue reports from various other repositories are also referenced in the Gitter chat rooms. Issues are informally discussed in Gitter whereas any information that has to be preserved is discussed in the issue tracker comments thread. We found out that in some projects up to 75% of the issues filed in the project's GitHub issue tracker are referenced in the Gitter chat rooms. However, for some projects such as `magento2` this percentage is very small, i.e., less than 1%. This suggests that some OSS developers rely on Gitter for collaboration and communication activities related to their project. The limited reliance on the other hand, in some cases, could be due to the presence of multiple chat rooms for a single project, e.g., across different platforms such as IRC or Discord.

6.3. ***Gitter allows real-time communication***

The obvious advantage of Gitter is its real-time communication feature which makes communication in Gitter quicker. Moreover, due to the informal style of chat communication [14], developers and end-users find it easy to comment on issues due to which feedback becomes faster. End-users are also encouraged by the quicker answers to support questions and choose to discuss issues in the chat rooms. Further, teams can discuss implementation details and reach common ground about design decisions, all without overloading the issue tracker threads. Maintainers make announcements regarding issues and provide updates that quickly reach everyone through a single channel.

Table 8: Spearman correlation between the number of collaborators and contributors involved in issue discussions in Gitter chat rooms and the issue resolution time. The bold values indicate statistically significant correlations.

Chat Room	**rho**	**p-value**
amber	0.010	9.18E-01
angular	0.077	**2.56E-04**
appium	0.012	8.96E-01
aws-sdk-go	-0.213	4.85E-01
bootstrap	-0.270	4.51E-01
corefx	0.072	4.41E-01
deeplearning4j	0.052	1.34E-01
gulp	-0.167	3.60E-01
JTAppleCalendar	0.520	**1.08E-06**
magento2	-0.136	5.67E-01
mailboxer	0.010	9.18E-01
material-design-lite	0.077	**2.56E-04**
meteor	0.012	8.96E-01
MonoGame	-0.213	4.85E-01
open-event-android	-0.270	4.51E-01
Perfect	0.072	4.41E-01
react-starter-kit	0.052	1.34E-01
scala-js	-0.167	3.60E-01
shuup	0.520	**1.08E-06**
theodinproject	-0.136	5.67E-01
typeScript	0.010	9.18E-01
vue	0.077	**2.56E-04**
webdriverio	0.012	8.96E-01
zipkin	-0.213	4.85E-01

Our understanding is that despite the availability of a dedicated GitHub issue tracker, developers and project collaborators find it comfortable to freely express their thoughts and opinions in chat messages, propose various alternatives to tackle a problem, argue about the possible implementations and collaboratively reach a final decision [18, 2]. An interesting direction for future work is to analyze the difference among the content, topics and style of issue report discussions in chat rooms and issue trackers.

6.4. ***Gitter helps to keep the issue tracker free of false issue reports***

We observed during manual annotation of issues that every time a problem is brought to Gitter, developers validate it first. In case it is a minor developer error, help is provided locally. This explains why a large number of issue report references on Gitter were attributed to *Technical support* and *GitHub issue tracker activity*. We also observed that sometimes issues raised as bugs are not bugs so Gitter allows core developers to check if the bug is valid before it reaches the issue tracker. Since new issue reports are only created against confirmed bugs, it helps keep the GitHub issue tracker free of false bug reports as well as useless discussions. Consequently, project maintainers are not unnecessarily pressurized due to an issue list that is overloaded with things that are not bugs.

6.5. ***Gitter helps to improve the triaging process***

Triaging is the process of deciding what to do with a newly submitted issue report, and it consumes an increasing amount of resources in large open-source projects [12]. Traditionally, an issue is considered by developers for resolution only after the triage [50]. As mentioned earlier, Gitter facilitates the triaging process whereby an issue report is only initiated in the tracker if Gitter participants consider it worthy of resolution. Sometimes the issue is promptly assigned to a project contributor who is a chat room participant and is willing to fix the issue. In this way, Gitter "preserves time of developers" for other issues, and also presumably eliminates delays in the triage process. An additional benefit of bringing an issue on Gitter is that it prevents issue creation against the wrong project or in the incorrect GitHub repository.

In summary, due to its unique features Gitter complements other sources of communication among project teams just like IRC and mailing lists were complementary to each other [19]. Open-source project stakeholders undergo extensive discussions in Gitter chat rooms after which the information that has to be preserved is brought to the issue trackers. Tools can be devised to identify, extract and transfer the necessary information to the issues trackers. Future work should also leverage this fragmented information across different channels to establish traceability links between various artifacts [51].

6.6. Study Implications

Our study is the first to investigate how and why issue reports are referenced on Gitter. We extracted the following implications and future research directions for researchers of issue reports and practitioners.

Gitter developers should consider structuring the Gitter data and adding more features. Currently, Gitter only allows referring an issue in chat but developers should consider introducing more features to support issue management, e.g., features or options that allow adding contextual information, reproducibility details, answering an issue comment, or, proposing a potential solution. **Practitioners should also develop tools to extract useful information from the Gitter data.** Such tools can perform automated analysis and summarization of discussions in the chat rooms and enable software developers to gain knowledge and discover information which might not be available on other channels. As an example, any discussions related to issues can be identified and later exported to the issue trackers.

Researchers of the process of resolving issue reports should include Gitter data in future studies. Due to the large number of issue report references and discussions around them on Gitter, researchers studying issue reports should include Gitter data in their studies since additional information regarding the issue resolution process can be found in there. Moreover, due to the repeated references to issue reports for *Technical help*, researchers could potentially leverage these references to design tools that can identify hard to resolve issues or issues that may not be resolved in the future fairly early in the issue management cycle. We hypothesize that identifying issues that may not be addressed in the future, could help developers to focus on critical issues, thereby improving the issue management process.

Practitioners should develop tools for automatic bug triaging by leveraging crowdsourced Gitter data to facilitate issue management process. Similarly bug triagers can be trained based on bug assignment or fix information available in Gitter chat rooms in the form of natural language text. Such tools when incorporated into current issue trackers will make the issue handling and resolution process more efficient.

Researchers should study how bots can be integrated on Gitter to support developers. Although in some chat rooms developers use bot integrations to support their work, the use of bots in the Gitter chat rooms is not as prevalent as we anticipated, given the fact that Gitter was designed to support bot integration. We observed that project developers carry out repetitive tasks on a daily basis, e.g., directing participants to the right resources such as PRs, WONTFIX issues or documentation. Our study indicates the need of bots integrated with the GitHub repositories and Gitter chat rooms to automate the ordinary tasks. In the context of this study, bots can be used to remind developers about long-standing issues, notify the appropriate team members when errors occur, auto-merge pull requests, link experts with novices, and answer user questions. Large development teams can be supported through dashboards showing different views of project activity to managers and developers, such as the opening and resolution of issues, execution of tests and merging of pull requests.

Researchers should study how the timing of referencing issue reports can contribute to the resolution process. Referencing issue reports on Gitter is a common practice which confirms the usefulness of Gitter. However, many issues are referenced on Gitter long after their creation, while the resolution likelihood of such issues increases after their reference on Gitter. Prior work shows that it is beneficial to attract attention to an issue report early on, e.g., by proposing a bounty [52]. These observations suggest that it can be beneficial to attract attention to an issue make report early on as well on Gitter. Future studies should investigate how the timing of referencing an issue is related to its resolution likelihood.

7. Threats to Validity

For RQ1, we relied on automatic similarity detection algorithms, but due to the lack of ground truth data we could not verify the reliability of the alias resolution process. However, one of the authors did a manual inspection of the aliases to decide a final set of identifiers which were also validated by another author.

Further, we manually identified the purpose of referencing issue reports and their discussions. To reduce bias due to human judgment, 2 authors finalized the categories by independently labelling the same set of issue discussions. The inter-rater agreement value was good enough for a single author to proceed with the rest of the labelling. However, chances of error due to human judgement exist which affects the internal validity of our study.

In RQ3, the conclusion validity was affected by multiple Mann-whitney comparisons. To alleviate the threat, we used Bonferroni corrected p-values. We also bootstrapped the issue resolution time 1,000 times and compared the distributions. Moreover, while comparing the Gitter issues with Random issues we controlled only four confounding factors which were found to be the top predictors of issue resolution time by the previous studies. Despite that our conclusions may be affected by other factors that we are not aware of.

As for the generalizability of this study, even though there are several types of rooms in Gitter such as organisational and user rooms, we only considered chat rooms that represent a GitHub repository. These repositories host open source projects and therefore, the results of our study will only be applicable in similar contexts. Future studies shall investigate how our results extend to other types of projects such as proprietary ones.

8. Conclusion

This paper conducts an empirical study of 14,096 issue report references in 24 developer chat rooms on Gitter, an open-source platform for hosting chat rooms that are directly coupled with GitHub projects. The most important findings of our study are:

- Comments and discussions over *GitHub issue tracker activity* contribute the highest percentage of issue report references after *technical support*.
- Issues that are referenced in the chat rooms have a longer resolution time compared to issues that are never referenced in Gitter.
- The number of comments on an issue in its GitHub issue tracker are slightly higher after an issue is referenced on Gitter than before it.
- The resolution likelihood of long-standing issues that are referenced on Gitter is higher than of those that are never referenced.

Our study shows that Gitter chat rooms are a rich data source for information about the issue resolution process in open source system. Therefore, we recommend that researchers of issue reports include this data source in their future studies.

References

[1] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, G. Antoniol, How developers' collaborations identified from different sources tell us about code changes, in: 2014 IEEE International Conference on Software Maintenance and Evolution, IEEE, 2014, pp. 251–260.

[2] E. Shihab, Z. M. Jiang, A. E. Hassan, Studying the use of developer IRC meetings in open source projects, in: 2009 IEEE International Conference on Software Maintenance, IEEE, 2009, pp. 147–156.

[3] B. Lin, A. Zagalsky, M.-A. Storey, 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 Companion, ACM, 2016, pp. 333–336.

[4] L. Dabbish, C. Stuart, J. Tsay, J. Herbsleb, Social coding in GitHub: transparency and collaboration in an open software repository, in: Proceedings of the ACM 2012 conference on computer supported cooperative work, ACM, 2012, pp. 1277–1286.

[5] H. Jiang, J. Zhang, Z. Ren, T. Zhang, An unsupervised approach for discovering relevant tutorial fragments for APIs, in: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, 2017, pp. 38–48.

[6] G. Petrosyan, M. P. Robillard, R. De Mori, Discovering information explaining API types using text classification, in: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, 2015, pp. 869–879.

[7] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, G. Canfora, Mining source code descriptions from developer communications, in: 2012 20th IEEE International Conference on Program Comprehension (ICPC), IEEE, 2012, pp. 63–72.

[8] E. Wong, J. Yang, L. Tan, Autocomment: Mining question and answer sites for automatic comment generation, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2013, pp. 562–567.

[9] L. B. de Souza, E. C. Campos, M. d. A. Maia, Ranking crowd knowledge to assist software development, in: Proceedings of the 22nd International Conference on Program Comprehension, ACM, 2014, pp. 72–82.

[10] V. Amintabar, A. Heydarnoori, M. Ghafari, ExceptionTracer: a solution recommender for exceptions in an integrated development environment, in: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, IEEE Press, 2015, pp. 299–302.

[11] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, M. Lanza, Mining StackOverflow to turn the IDE into a self-confident programming prompter, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 102–111.

[12] A. S. Badashian, A. Hindle, E. Stroulia, Crowdsourced bug triaging, in: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2015, pp. 506–510.

[13] C. Chen, Z. Xing, X. Wang, Unsupervised software-specific morphological forms inference from informal discussions, in: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, 2017, pp. 450–461.

[14] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, A. Zagalsky, The (r) evolution of social media in software engineering, in: Proceedings of the on Future of Software Engineering, ACM, 2014, pp. 100–116.

[15] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, 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.

[16] S. A. Chowdhury, A. Hindle, Mining StackOverflow to filter out off-topic IRC discussion, in: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, 2015, pp. 422–425.

[17] V. Käfer, D. Graziotin, I. Bogicevic, S. Wagner, J. Ramadani, Communication in open-source projects-end of the e-mail era?, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 242–243.

[18] R. Alkadhi, T. Lata, E. Guzmany, B. Bruegge, Rationale in development chat messages: an exploratory study, in: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 436–446.

[19] L. Yu, S. Ramaswamy, A. Mishra, D. Mishra, Communications in global software development: an empirical study using GTK+ OSS repository, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2011, pp. 218–227.

[20] C. Bird, A. Gourley, P. Devanbu, M. Gertz, A. Swaminathan, Mining email social networks, in: Proceedings of the 2006 international workshop on Mining software repositories, ACM, 2006, pp. 137–143.

[21] R. Alkadhi, J. O. Johanssen, E. Guzman, B. Bruegge, React: an approach for capturing rationale in chat messages, in: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2017, pp. 175–180.

[22] S. Baltes, C. Treude, S. Diehl, Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets, in: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE, 2019, pp. 191–194.

[23] N. Bettenburg, A. E. Hassan, Studying the impact of social structures on software quality, in: 2010 IEEE 18th International Conference on Program Comprehension, IEEE, 2010, pp. 124–133.

[24] R. Abreu, R. Premraj, How developer communication frequency relates to bug introducing changes, in: Proceedings of the joint international and

annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, ACM, 2009, pp. 153–158.

[25] T. Wolf, A. Schroter, D. Damian, T. Nguyen, Predicting build failures using social network analysis on developer communication, in: Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, 2009, pp. 1–11.

[26] A. Bacchelli, M. D'Ambros, M. Lanza, Are popular classes more defect prone?, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2010, pp. 59–73.

[27] M. Ortu, G. Destefanis, M. Kassab, S. Counsell, M. Marchesi, R. Tonelli, Would you mind fixing this issue?, in: International Conference on Agile Software Development, Springer, 2015, pp. 129–140.

[28] Y. Zhao, F. Zhang, E. Shihab, Y. Zou, A. E. Hassan, How are discussions associated with bug reworking?: An empirical study on open source projects, in: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2016, p. 21.

[29] G. Navarro, A guided tour to approximate string matching, ACM computing surveys (CSUR) 33 (1) (2001) 31–88.

[30] A. Mockus, R. T. Fielding, J. Herbsleb, A case study of open source software development: the Apache server, in: Proceedings of the 22nd international conference on Software engineering, 2000, pp. 263–272.

[31] R. V. Krejcie, D. W. Morgan, Determining sample size for research activities, Educational and psychological measurement 30 (3) (1970) 607–610.

[32] H. Hata, C. Treude, R. G. Kula, T. Ishio, 9.6 million links in source code comments: purpose, evolution, and decay, in: Proceedings of the 41st International Conference on Software Engineering, IEEE Press, 2019, pp. 1211–1221.

[33] H. Hu, S. Wang, C.-P. Bezemer, A. E. Hassan, Studying the consistency of star ratings and reviews of popular free hybrid Android and iOS apps, Empirical Software Engineering 24 (1) (2019) 7–32.

[34] S. Hassan, C.-P. Bezemer, A. E. Hassan, Studying bad updates of top free-to-download apps in the Google Play Store, IEEE Transactions on Software Engineering.

[35] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, D. Lo, Categorizing the content of github readme files, Empirical Software Engineering 24 (3) (2019) 1296–1327.

[36] J. Corbin, A. Strauss, Basics of qualitative research: Techniques and procedures for developing grounded theory, Sage publications, 2014.

[37] A. J. Viera, J. M. Garrett, et al., Understanding interobserver agreement: the Kappa statistic, Fam med 37 (5) (2005) 360–363.

[38] J. Park, M. Kim, B. Ray, D.-H. Bae, An empirical study of supplementary bug fixes, in: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, IEEE Press, 2012, pp. 40–49.

[39] F. J. Massey Jr, The Kolmogorov-Smirnov test for goodness of fit, Journal of the American statistical Association 46 (253) (1951) 68–78.

[40] M. Neuhäuser, Wilcoxon–Mann–Whitney test, in: M. Lovric (Ed.), International Encyclopedia of Statistical Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 1656–1658. doi:10.1007/978-3-642-04898-2_615.
URL https://doi.org/10.1007/978-3-642-04898-2_615

[41] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, L. Devine, Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's-d indices the most appropriate choices, in: annual meeting of the Southern Association for Institutional Research, Citeseer, 2006, pp. 1–51.

[42] P. J. Guo, T. Zimmermann, N. Nagappan, B. Murphy, Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows, in: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1, 2010, pp. 495–504.

[43] R. Kikas, M. Dumas, D. Pfahl, Using dynamic and contextual features to predict issue lifetime in GitHub projects, in: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), IEEE, 2016, pp. 291–302.

[44] H. Zhang, L. Gong, S. Versteeg, Predicting bug-fixing time: an empirical study of commercial software projects, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 1042–1051.

[45] E. Giger, M. Pinzger, H. Gall, Predicting the fix time of bugs, in: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, 2010, pp. 52–56.

[46] D. Arya, W. Wang, J. L. Guo, J. Cheng, Analysis and detection of information types of open source software issue discussions, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 454–464.

[47] C. Weiss, R. Premraj, T. Zimmermann, A. Zeller, How long will it take to fix this bug?, in: Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007), IEEE, 2007, pp. 1–1.

[48] A. Di Sorbo, J. Spillner, G. Canfora, S. Panichella, "won't we fix this issue?" qualitative characterization and automated identification of Wontfix issues on GitHub, arXiv preprint arXiv:1904.02414.

[49] J. H. Zar, Significance testing of the Spearman rank correlation coefficient, Journal of the American Statistical Association 67 (339) (1972) 578–580.

[50] H. Hu, H. Zhang, J. Xuan, W. Sun, Effective bug triage based on historical bug-fix information, in: 2014 IEEE 25th International Symposium on Software Reliability Engineering, IEEE, 2014, pp. 122–132.

[51] A. Hindle, C. Bird, T. Zimmermann, N. Nagappan, Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers?, in: 2012 28th IEEE International Conference on Software Maintenance (ICSM), IEEE, 2012, pp. 243–252.

[52] J. Zhou, S. Wang, C.-P. Bezemer, Y. Zou, A. E. Hassan, Bounties in open source development on GitHub: A case study of BountySource bounties, IEEE Transactions on Software Engineering.