2018 25th Asia-Pacific Software Engineering Conference (APSEC)

Multi-discussing Across Issues in GitHub: A Preliminary Study

Dongyang Hu, Tao Wang, Junsheng Chang, Gang Yin, Yang Zhang*
Science and Technology on Parallel and Distributed Processing Laboratory
National University of Defense Technology, Changsha, China
{*hudongyang17,taowang2005,cjs7908,jack_nudt,yangzhang15*}*@nudt.edu.cn*

***Abstract*—Social coding sites like GitHub has enabled developers to easily contribute their comments on multiple issues and switch their discussion *between* issues, *i.e.*, multi-discussing. Discussing multiple issues simultaneously may enhance the work efficiency of developers. However, multi-discussing also relies on developers' rationally allocating their time and focus, which may bring different influence to the resolution of issues. Therefore, investigating how multi-discussing affects the issue resolution is a meaningful research question which can help developers understand the benefits and limitations when they switch their discussion between issues. In this paper, we present a preliminary study of the impact of multi-discussing on issue resolution in GitHub projects, by using quantitative methods. First, we collect and analyzed data from 631 GitHub projects to explore how multi-discussing affects the average resolution latency of project issues. Further, we develop method for measuring the rate and breadth of a developers' discussion-switching behavior, and we use regression modeling to study how discussion-switching affects the single issue resolution latency. We find that multi-discussing is a common behavior of developers in GitHub projects. Also, multi-discussing is associated with shorter average issue resolution latency of project. However, during a single issue resolution, more participants' discussion-switching tend to bring longer issue resolution latency. Our study motivates the need for further research on the multi-discussing.**

***Keywords*-Multi-discussing; GitHub; Issue resolution;**

I. Introduction

GitHub, one of the premier open source development sites, has hosted more than 85M repositories as of June 2018[1]. By implementing the concept of *social coding*, GitHub creates a developer-friendly environment that enables developers to network, collaborate, and promote their projects [1]. Such social coding also leads to enhanced issue resolution process, in which developers can easily participate in reporting and discussing issues. As a result, it is not uncommon to find that developers contribute their comments on multiple issues and switch their discussion between issues. In our study, we define it as *multi-discussing*.

Multi-discussing, as a common phenomenon in today's era of agile software development, can be seen as a kind of multitasking. Multitasking aims to optimize human resource allocation and reprioritize tasks dynamically, which is the ability for developers to stop dealing with a task, switch to another task, and return to its original mission, as needed or as scheduled [2]. Recently, many studies have explored the multitasking in GitHub Projects [3], [4], [5]. However, their main goals were to investigate how developers do the multitasking between projects, *i.e.*, at a project-level. To the best of our knowledge, no existing research has been conducted on how developers do the multi-discussing between issues, *i.e.*, at a issue-level. On the one hand, multi-discussing may enhance the work efficiency of developers, helping them discuss multiple issues simultaneously. On the other hand, multi-discussing relies on developers' rationally allocating their time and focus, which may vary. Different strategies and focus of multi-discussing may bring different influences to the issue resolution process. Is the multi-discussing a common behavior in the developer community? How does multi-discussing affect the issue resolution? Answering those questions can help developers understand the benefits and limitations when they switch their discussion and focus between issues.

In this paper, we perform a preliminary study of multi-discussing in GitHub issues using quantitative analysis methods. In our data collection process, we obtain the issue data from 631 large projects (each of them contains 3,000+ issues). We investigate the basic usage of multi-discussing by computing the percentage of issues with multi-discussing participants. Next, to explore how multi-discussing affects the issue resolution, we conduct our investigation on two levels, *i.e.*, project-level and issue-level. At the project-level, we seek to study how multi-discussing affects the total issue resolution of a project, by regression modeling the relationship between the multi-discussing percentage in a project and the average issue resolution latency of this project. At the issue-level, we aim to analyze how developers' discussion-switching behavior in an issue affects this issue's resolution, by developing the measurement of *discussion focus* and regression modeling the relationship between the discussion focus and the single issue resolution latency. Finally, we distill some implications for researchers and developers.

In summary, our findings are:

- On average, 65% of issues in our dataset contains the multi-discussing developers, but the percentage of different projects vary.
- Multi-discussing tends to bring benefit to the overall

*Corresponding author.
[1]https://github.com/about

issue resolution of project, *i.e.*, reducing the average issue resolution latency.

- Multi-discussing may have limitations within a single issue resolution, because more discussion-switching behavior is associated with longer issue resolution latency.

The structure of the remainder of the paper is as follows. In §II, we introduce some background knowledge and our research questions. Next, in §III, we elaborate our empirical study methodology. We present our study results in §IV. We discuss implications of our study and threats to validity in §V. We conclude and mention future work in §VI.

II. Preliminaries

First, we introduce some background knowledge related to our study and present our research questions.

A. Issue Resolution and Discussion

In GitHub, issue resolution is a complex process which has been studied by many researches [6], [7], [8]. Murgia *et al.* [6] investigated the influence of the maintenance type on issue resolution time. From GitHub's issue repository, they analyzed more than 14000 issue reports taken from 34 open source projects and classify them as corrective, adaptive, perfective or preventive maintenance to show that the issue resolution time depends on the maintenance type. Bissyandé *et al.* [7] addressed the need for answering such questions by performing an empirical study on a hundred thousands of open source projects. After filtering relevant trackers, their study used about 20,000 projects and they answered various research questions on the popularity and impact of issue trackers. Zhang *et al.* [8] presented a mixed methods study of the relationship between the practice of issue linking and issue resolution in the Rails ecosystem. They found that linking across projects will not retard issue resolution, and it is associated with more discussion.

Recently, some researches began to study how developers discuss in the issue resolution [9], [10], [11], [12], [13]. Tsay *et al.* [9] presented a study of how developers in open work environments evaluate and discuss pull requests, a primary method of contribution in GitHub, analyzing a sample of extended discussions around pull requests and interviews with GitHub developers. Modern software development is increasingly collaborative [10]. Open Source Software (OSS) are the bellwether; they support dynamic teams, with tools for code sharing, communication, and issue tracking. The success of an OSS project is reliant on team communication [11]. E.g., in issue discussions, individuals rely on rhetoric to argue their position, but also maintain technical relevancy. Rhetoric and technical language are on opposite ends of a language complexity spectrum: the former is stylistically natural; the latter is terse and concise. Issue discussions embody this duality [12], as developers use rhetoric to describe technical issues. The style mix in any discussion can define group culture and affect performance, e.g., issue resolution time may be longer if discussion is imprecise. Using GitHub, they studied issue discussions to understand whether project-specific language differences exist, and to what extent users conform to a language norm. Different from those prior studies, our work aims to investigate the impact of multi-discussing on issue resolution latency.

B. Developer Collaboration

Developer collaboration plays an important role in the issue resolution, which has been investigated in many studies [1], [14], [15], [16], [17], [18]. Dabbish *et al.* [1] have found that people make a surprisingly rich set of social inferences from the networked activity information in GitHub, such as inferring someone else's technical goals and vision when they edit code, or guessing which of several similar projects has the best chance of thriving in the long term. Users combine these inferences into effective strategies for coordinating work, advancing technical skills and managing their reputation. Thung *et al.* [14] identified influential developers and projects on this sub network of GitHub by using PageRank. Understanding how developers and projects are actually related to each other on a social coding site is the first step towards building tool supports to aid social programmers in performing their tasks more efficiently. Wu *et al.* [15] hypothesized that the social system of GitHub users was bound by system interactions such that contributing to similar code repositories would lead to users following one another on GitHub or vice versa. Using a quadratic assignment procedure (QAP) correlation, however, only a weak correlation among followship and production activities (code, issue, and wiki contributions) was found. Due to these preliminary findings, they described GitHub as a part of a larger ecosystem of developer interactions. Zhang *et al.* [16] conducted an exploratory study of the developer collaboration by investigating the social media tool @-mention in GitHub's pull-requests. Their study results showed that @-mention attracts more participants [17] and favors the solving process of issues by enlarging the visibility of issues and facilitating the developers' collaboration [18]. Our work also concerns the developer collaboration, but mainly focus on how developer collaborate in the issue discussion.

C. Multi-discussing

Multi-discussing can be seen as a kind of multitasking on the issue-level. Many researches have studied the multitasking and found that it has become increasingly common among knowledge workers [5], [19], [20]. While there is some disagreement between studies, it is generally believed that multitasking has non-monotonic(concave) effects on individual productivity [2], with plenty of theoretical evidence from psychology, management, and organizational behavior.

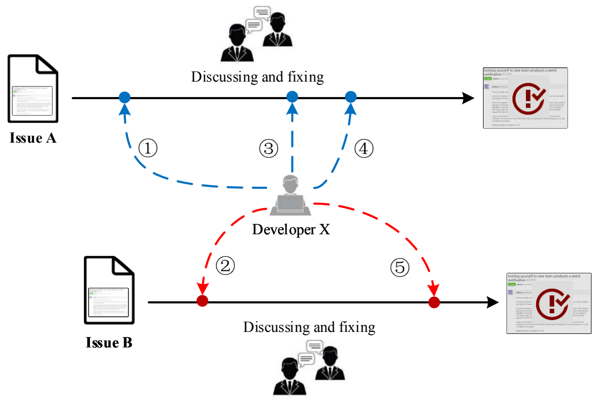


Figure 1: Overview of the multi-discussing process.

Multi-discussing across project issues is a focus switching behavior of developers. Figure 1 shows the overview of the multi-discussing process. During the resolution process of issue A, many developers may participate in and discuss, trying to fix this issue. In particular, developer X may participated in issue A's discussion first (①). Then, although issue A was not closed, developer X switched his/her focus to another issue, issue B, and contributed a comment in issue B's discussion (②). Next, although issue B was not closed, developer X moved to issue A's discussion again and contributed two comments (③ and ④). After that, developer X participated issue B's discussion again (⑤). From this discussion-switching process, we find that developer X switched his/her focus between issue A and issue B over time. Figure 2 gives an example of multi-discussing in Rails project. We find that developer *utilum* discussed on issue #32953 at May 23, 2018. Before issue #32953 was closed, developer *utilum* switched his/her focus on the issue #32994 and commented at May 26, 2018.

D. Research Questions

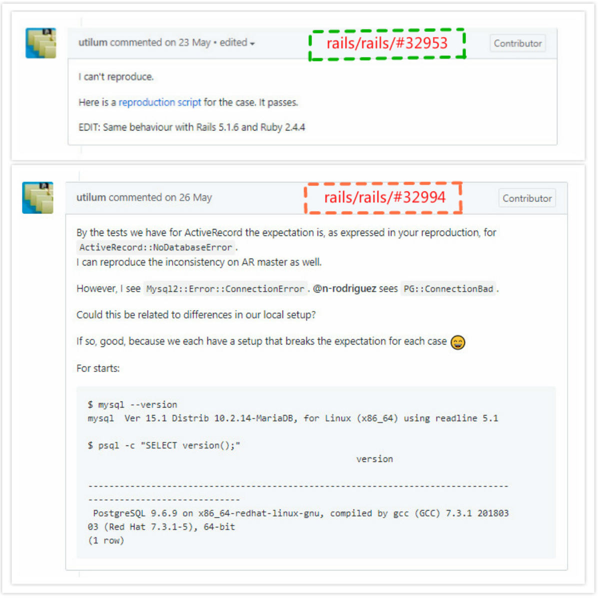
Although multi-discussing is an ordinary behavior of developers in practice, few research has been conducted on it. To fill this research gap, we aim to explore how developers do the multi-discussing and what the impact of multi-discussing on issue resolution. Specifically, to better understand the multi-discussing usage, we ask:

RQ1: *How many issues in which the participants do the multi-discussing?*

To answer this question, we collect issue data from 631 large GitHub projects and calculate the percentage of issues that have multi-discussing participants.

Next, we seek to study how multi-discussing affects the issue resolution. We ask the following two questions:

RQ2: *How does multi-discussing affect the overall issue resolution of a project?*



utilum commented on 23 May • edited ▾ rails/rails/#32953 Contributor

I can't reproduce.

Here is a reproduction script for the case. It passes.

EDIT: Same behaviour with Rails 5.1.6 and Ruby 2.4.4

utilum commented on 26 May rails/rails/#32994 Contributor

By the tests we have for ActiveRecord the expectation is, as expressed in your reproduction, for `ActiveRecord::NoDatabaseError`.
I can reproduce the inconsistency on AR master as well.

However, I see `Mysql2::Error::ConnectionError`. @n-rodriguez sees `PG::ConnectionBad`.

Could this be related to differences in our local setup?

If so, good, because we each have a setup that breaks the expectation for each case 😄

For starts:

```
$ mysql --version
mysql  Ver 15.1 Distrib 10.2.14-MariaDB, for Linux (x86_64) using readline 5.1

$ psql -c "SELECT version();"
                                                    version

---------------------------------------------------------------------------------
-----------------------------
 PostgreSQL 9.6.9 on x86_64-redhat-linux-gnu, compiled by gcc (GCC) 7.3.1 201803
03 (Red Hat 7.3.1-5), 64-bit
(1 row)
```

Figure 2: An example of multi-discussing in Rails project.

Answering this question, we develop the overall issue resolution model to investigate the relationship between the percentage of multi-discussing issues in a project and the average issue resolution latency of this project.

RQ3: *How does multi-discussing affect the resolution of a single issue?*

To answer this question, we select the issues from two large and famous projects, *i.e.*, Rails and TrinityCore, and build the single issue resolution models and developer focus models to study how developers' discussion-switching affects the single issue resolution latency.

III. Methodology

Here, we present the methodology of our study.

A. Data Set

We collect data from the population of open-source projects in GitHub. In this work, we obtain the data from GHTorrent [21]. It is an effort to create a scalable, offline mirror of data offered through the Github REST API.

In our study, to better investigate the multi-discussing in issues, we only select those large projects that have more than 3,000 issues in their historical data. Note that we only consider the general issues, we don't consider the pull requests in this study. Further, after removing the projects that were forked from other projects or have been deleted from GitHub, we collect the information of 631 projects for our next data collection process. And note that in our study, we only consider those developers who are not project members, *i.e.*, external contributors. We don't consider project members because they are very likely to

Table I: Aggregate statistics of the 631 projects.

Statistic	Mean	St.Dev.	Min	Median	Max
#Members	21.2	42.5	2.0	9.0	440.0
#Forks	2,275.0	5,629.0	3.00	1,034.0	93,397.0
#Stars	7,893.0	15,691.2	10.0	3,129.0	215,302.0
#Issues	6,649.0	3,567.3	3,484.0	5,259.0	18,614.0

do the multi-discussing in issues, *e.g.*, answering issue reporters' questions and discussing with external contributors. Table I shows the aggregate statistics of the 631 projects. On average, the number of issues in each project is 6,649 (median is 5,259; max is 18,614), and the number of forks is 2,275 (median is 1,034; max is 93,397). So the projects in our dataset are all large and famous projects.

Note that in our RQ3, we only choose two famous and large projects for our study, *i.e.*, Rails[2] and TrinityCore[3]. Because they are very famous and success on GitHub with a lot of stars and forks, and they are all active projects with a long life cycle, a number of experienced developers, and plenty of historical data.

B. Measuring the Discussion-switching Focus – D_{Focus}

In our RQ3, to explore how multi-discussing affects the single issue resolution latency, we develop a measurement, *i.e.*, D_{Focus}, to define the rate and breadth of developers' discussion-switching focus in a given week. In this work, we use the Teachman/Shannon entropy index [3], [22], to calculate the D_{Focus}:

$$D_{Focus} = \frac{1}{-\sum_{i=1}^{N} p_i \log_2 p_i + 1} \quad (1)$$

Considering a developer Y in the i'th week, p_i is the issue ID of issue that Y participated in, and N is the total number of issues that Y participated in during the i'th week. Thus, the higher the D_{Focus} value, the lower the developer's focus on the discussion of a single issue. For example, Figure 3 shows two examples of calculating the D_{Focus} between developer A and developer B. A and B discussed on issues both five times in a week. But these five times were discussed in different degrees of dispersion between A and B. A discussed on three different issues, *i.e.*, #17257, #19237, and #39044, and discussed in issue #17257 three times in a week. B discussed on four different issues, *i.e.*, #17257, #19237, #22688, and #39044, and his discussion of issues were relatively scattered in a week. So the D_{Focus} (the value is 0.73) of A is higher than the D_{Focus} (the value is 0.52) of B.

C. Regression Analysis

1) Overall issue resolution model: First, to find how multi-discussing affects the overall issue resolution of a project, we build the mixed-effects linear regression model (using packages `lme4` and `lmerTest` in R), *i.e.*, *overall issue resolution model* (**Model-1**), with the random-effect factor *programming language*. The random effect allows us to avoid modeling each project language separately, which would use up degrees of freedom unnecessarily, but still capture project language variability in the response. All other variables are modeled as fixed effects. The dependent variable (outcome) of this model is:

- ***avgIssueLatency***: the average issue resolution latency in a project, in minutes, as a proxy for the overall issue resolution speed. In our study, we only consider the issue latency between the issue creation to its first closing date.

The control and independent variables of this model are described as follows:

- ***mdPerc***: the percentage of issues that contain the multi-discussing developers, as a proxy for the multi-discussing usage of the project.
- ***nIssues***: total number of issues in the project, as a proxy for the overall workload of the project.
- ***nMembers***: total number of members in the project, as a proxy for the human resource of the project.
- ***nForks***: total number of forks in the project, as a proxy for the relationship network of the project.
- ***nStars***: total number of stars in the project, as a proxy for the popularity of the project.

2) Single issue resolution models: Second, to study how multi-discussing affects the single issue resolution, we build two multiple linear regression models (using package `lm` in R), *i.e.*, *Rails single issue resolution latency model* (**Model-2**) and *TrinityCore single issue resolution latency model* (**Model-3**). The dependent variables of the two models are:

- ***singleIssueLatency***: the issue resolution latency of a single issue, in minutes, as a proxy for single issue resolution speed.

The independent variables of the two models are described as follows:

- ***nComments***: total number of comments in this issue, as a proxy for the discussion length of this issue.

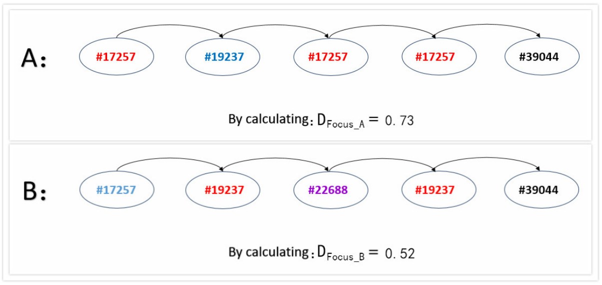

A:
#17257
#19237
#17257
#17257
#39044
By calculating: $D_{Focus\_A} = 0.73$
B:
#17257
#19237
#22688
#19237
#39044
By calculating: $D_{Focus\_B} = 0.52$

Figure 3: Two examples of calculating the D_{Focus}.

[2]https://github.com/rails/rails
[3]https://github.com/TrinityCore/TrinityCore

- ***nDevelopers***: total number of developers that participated in this issue, as a proxy for the human effort of this issue.
- ***subMemberTag***: Binary, True if the issue submitter is one of the project members.
- ***hasLabelTag***: Binary, True if this issue has at least one label.
- ***textLen***: total words length of the issue text (title and description), as a proxy for issue complexity. Longer descriptions may indicate higher complexity of issue or better documentation [23].
- ***multi-discussingTag***: Binary, True if this issue has at least one multi-discussing participant.

3) Developer focus models: Third, to explore how developers' discussion focus effects the issue resolution, we build two models, *i.e.*, *Rails developer focus model* (**Model-4**) and *TrinityCore developer focus model* (**Model-5**). The dependent variables of the two models are still the ***singleIssueLatency***, as described before. The independent variables of the two models are ***nComments***, ***nDevelopers***, ***subMemberTag***, ***hasLabelTag***, ***textLen***, and

- ***avgFocus***: average D_{Focus} value of all developers discussed in this issue. Note that we do not consider the project members.

In our models, where necessary we log-transform dependent variables to stabilize their variance and reduce heteroscedasticity [24]. We also remove the top 2% of the data for highly-skewed variables, to control outliers and improve model robustness, in line with best practices [25]. The variance inflation factors, which measure multicollinearity of the set of predictors in our models, are safe, below 3. For each model variable, we report its coefficients, standard error, significance level, and sum of squares (via ANOVA analyses). We consider the such coefficients noteworthy if they were statistically significant at p<0.05. The mixed-effects linear regression model fit is evaluated using a marginal (R^2_m) and a conditional (R^2_c) coefficient of determination for generalized mixed-effects models [26]. R^2_m describes the proportion of variance explained by the fixed effects alone, and R^2_c describes the proportion of variance explained by the fixed and random effects together.

IV. Study Results

In this section, we report our study results that respond to the research questions.

A. RQ1: How many issues in which the participants do the multi-discussing?

To understand the basic usage of multi-discussing in developer community, we calculate the percentage of issues than contains at least one multi-discussing developers in the 631 projects. During one issue's lifetime, *i.e.*, from its creation to its first closing date, if at least one of its participants also discussed in other issue, we call this issue as *multi-discussing issue*, and mark the multi-discussing tag of this issue as 1, otherwise as 0. After marking each issue of a project, we can calculate the percentage of multi-discussing issues of this project. Figure 4 shows the boxplot of the percentage of multi-discussing issues among the 631 projects. On average, the percentage of multi-discussing issues is 0.650 (median of 0.598). However, we find that the percentage of multi-discussing issues in different projects vary. Thus, we find that,

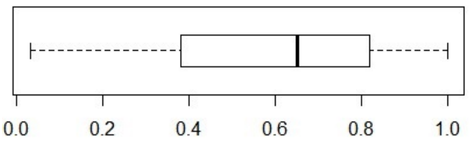


Figure 4: The distribution of the percentage of multi-discussing issues in the 631 project.

On average, in 65% of project issues, the participants did the multi-discussing in their issue resolution.

B. RQ2: How does multi-discussing affect the overall issue resolution of a project?

To study how multi-discussing affects the overall issue resolution of a project, we calculate the issue resolution latency of each issue in the 631 projects. Next, for each project, we calculate its average issue resolution latency. In RQ1, we have obtained the percentage of multi-discussing issues of each project. To compare the differences of average issue resolution latency between different multi-discussing percentages, we mark the value of percentage from 0 to 10% as 0.1, from 10% to 20% as 0.2, from 20% to 30% as 0.3, and so on. Figure 5 shows the boxplot of the comparison results. We can find that although from 0.1 to 0.3, the average issue resolution latency increases, from 0.3 to 0.9, the average issue resolution latency is gradually decreasing. It shows that more multi-discussing in a project seem to bring more benefit to the overall issue resolution.

In order to better explore the effect of multi-discussing on the overall issue resolution of a project, we build the regression model, *i.e.*, *Overall issue resolution model* (**Model-1**), as described in Sec. III-C. Table II shows the summary of this model. The fixed-effects part of the model explained R^2_m=0.40 of the deviance. In total, the model explained R^2_c=0.42 of the deviance with the random effect.

As expected, the factor ***mdPerc*** has a significant negative effect on the average issue resolution latency. 10% increase to ***mdPerc*** has the effect of decreasing the average issue resolution latency by 1-$e^{-0.6390}$ =47.2%, holding all other variables constant. Thus, we find that,

For the overall issue resolution, more multi-discussing tend to bring shorter average issue resolution latency.

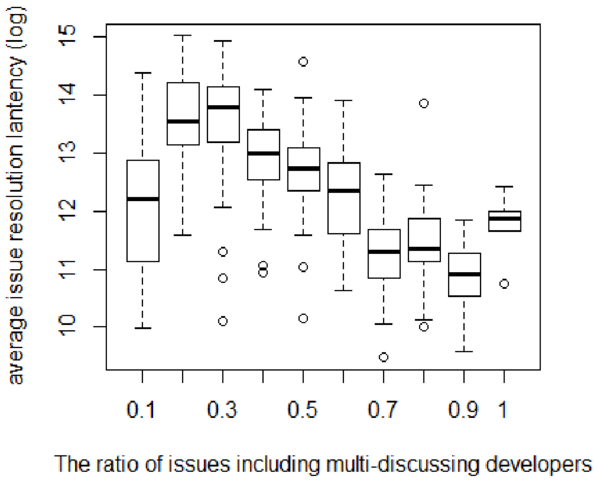


Figure 5: The average issue resolution lantency of projects.

Table II: Overall issue resolution model. Resp.: log(avgIssueLatency), R^2_m=0.40, R^2_c=0.42.

	Estimate (Error)	Sum Sq.
(Intercept)	0.0025 (0.0442)	
mdPerc	-0.6390 (0.0410)***	143.63***
nIssues	0.0579 (0.0432).	1.06
nMembers	-0.0637 (0.0413).	1.41
nForks	-0.0353 (0.0449)	0.36*
nStars	-0.0866 (0.0450)*	2.19*

***$p < 0.001$, **$p < 0.01$, *$p < 0.05$, .$p < 0.01$

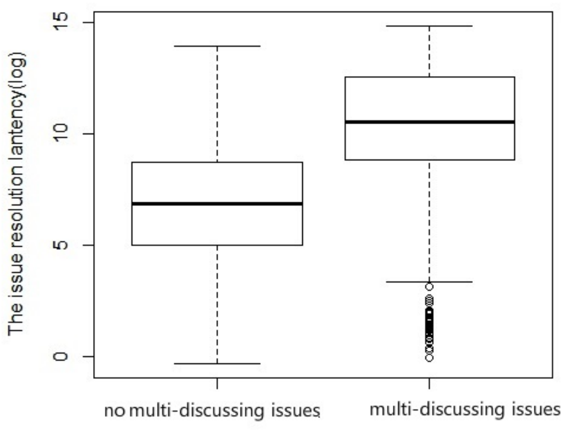


Figure 6: The difference of the issue resolution latency between multi-discussing issues and no multi-discussing issues in the Rails project.

Table III: Rails single issue resolution latency model. Resp.: log(singleIssueLatency), R^2=0.19.

	Estimate (Error)	Sum Sq.
(Intercept)	-0.5120 (0.0195)***	
multi-discussingTag True	0.5011 (0.0234)***	718.60***
textLen	-0.0038 (0.0111)	0.50
nDevelopers	0.1825 (0.0138)***	265.90***
nComments	0.0005 (0.0137)	0.40
subMemberTag Ture	-0.1309 (0.0731).	0.00
hasLabelTag Ture	0.4431 (0.0233)***	290.20***

***$p < 0.001$, **$p < 0.01$, *$p < 0.05$, .$p < 0.01$

C. RQ3: How does multi-discussing affect the resolution of a single issue?

1) Study-1: To study how multi-discussing affects the single issue resolution, we select two famous and large projects in our dataset, Rails and TrinityCore. Based on the issues of the two projects, we first compare the difference of issue resolution latency between the multi-discussing issues and those issues without multi-discussing participants. From statistics, there are 4,207 (65.4%) issues including multi-discussing developers in Rails project and 7,208 (78.5%) issues including multi-discussing developers in TrinityCore project. The other issues in these two projects have no multi-discussing developers. In particular, we ask:

RQ3-1: Is there a difference of single issue resolution latency between multi-discussing issues and those issues without multi-discussing?

A. Rails issues. Figure 6 shows the boxplots of issue resolution latency difference between multi-discussing issues and no multi-discussing issues. On average, the issue resolution latency of each multi-discussing issue in Rails project is 131.1 days (median of 25.9 days), while the issue resolution latency of each no multi-discussing issues is 16.4 days (median of 1.9 days). The Whitney-Mann-Wilcoxon (WMW) test shows that the difference between those two datasets is significant (W=1.9$\times$10^6; p<2.2$\times$10^{-16}). This analysis does not consider many of the possible confounds, so for more precise findings we turn to the multiple regression analysis. Table III shows the results of Rails issue resolution latency model (**Model-2**). The adjusted fit of this model is R^2=0.19.

As expected, the factor ***multi-discussingTag*** has a significant positive effect on the single issue resolution latency (56.3% of the variance explained). A one-unit increase in ***multi-discussingTag*** (from FALSE, encoded as 0, to TRUE, encoded as 1), results in the increase in the issue resolution latency of $e^{0.5011}$-1 =65.1%, holding all other variables constant. The factor ***nDevelopers*** has a significant positive effect on the single issue resolution latency (20.8% of the variance explained). 10% increase to ***nDevelopers*** has the effect of increasing the issue resolution latency outcome by 20.0%, holding all other variables constant. More participants may indicate that issue is a complex problem which need more developers to resolve. The factor ***hasLabelTag*** has a significant positive effect on the single issue resolution latency (22.8% of the variance explained). A one-unit increase in ***hasLabelTag***, results in the increase in the issue resolution

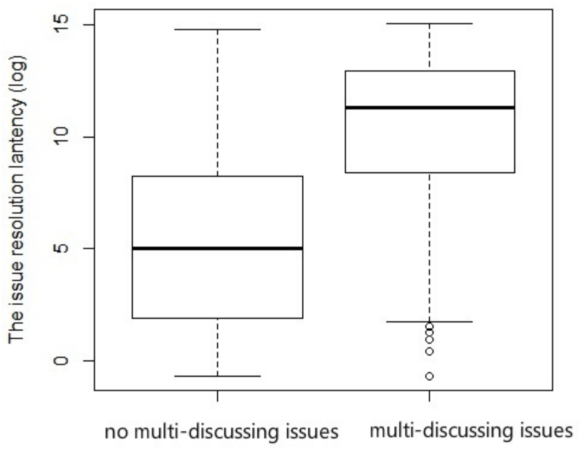


Figure 7: The difference of the issue resolution latency between multi-discussing issues and no multi-discussing issues in the TrinityCore project.

Table IV: TrinityCore single issue resolution latency model. Resp.: log(singleIssueLatency), R^2=0.23.

	Estimate (Error)	Sum Sq.
(Intercept)	-0.3263 (0.0137)***	
multi-discussingTag Ture	0.1335 (0.0190)***	1489.50***
textLen	-0.0068 (0.0074)	1.80
nDevelopers	0.3395 (0.0092)***	1133.20***
nComments	-0.0110 (0.0074)	5.40**
subMemberTag Ture	0.1903 (0.0232)***	146.70***
hasLabelTag Ture	0.4044 (0.0164)***	464.80***

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

latency of 55.8%, holding all other variables constant. Issue with labels may be difficult problem, so it need longer time to resolve. According to the results, we find that, in Rails project, multi-discussing issues tend to have longer issue resolution latency than those issues without multi-discussing developers.

B. TrinityCore issues. Figure 7 shows the boxplots of issue resolution latency difference between multi-discussing issues and no multi-discussing issues in the TrinityCore project. On average, the issue resolution latency of each multi-discussing issue in TrinityCore project is 231.1 days (median of 54.6 days), and the issue resolution latency of each no multi-discussing issue in TrinityCore project is 39.8 days (median of 1.4 days). The WMW test shows that the difference between two datasets is significant (W=8.4$\times$10^6; p<2.2$\times$10^{-16}). Table IV shows the results of the TrinityCore single issue resolution model (**Model-3**). The adjusted fit of this model is R^2=0.23.

As expected, The control ***multi-discussingTag*** is significant and has the most sizeable and positive effect on the single issue resolution latency. A one-unit increase in ***multi-discussingTag***, results in the increase in the issue resolution lantency of 14.3%, holding all other variables constant. The factor ***nDevelopers*** has a significant positive effect on the single issue resolution lantency (35.0% of the variance explained). 10% increase to ***nDevelopers*** has the effect of increasing the issue resolution latency outcome by 40.4%, holding all other variables constant. Like in Rails project, more participants may indicate that issue is a complex problem which need more developers to resolve. Not like in Rails project, The factor ***subMemberTag*** has a positive effect on the single issue resolution lantency (4.5% of the variance explained) in TrinityCore project. A one-unit increase in ***subMemberTag***, results in the increase in the issue resolution latency of 20.9%, holding all other variables constant. Issue submitted by project member may be more difficult, so it need longer time to resolve. The factor ***hasLabelTag*** also has a significant positive effect on the single issue resolution lantency (14.3% of the variance explained). A one-unit increase in ***hasLabelTag***, results in the increase in the issue resolution latency of 49.8%, holding all other variables constant.

According to the results, we find that, in the TrinityCore project, multi-discussing issues also tend to have longer issue resolution latency than those issues without multi-discussing. Thus, we find that,

Both in Rails and TrinityCore projects, multi-discussing issues tend to have longer issue resolution latency than those issues without multi-discussing.

2) Study-2: During an issue's resolution, there may exist many participants. Some of them may perform the multi-discussing, some of them may not, which may bring different effects to the issue resolution latency. To measure the focus of a developer when switching his/her discussion between issues, we use the D_{Focus}, as defined in Sec. III-B. For a single issue, we want to investigate how the average focus of all participants affects the single issue resolution latency. So we ask:

RQ3-2: How does the average D_{Focus} of an issue affect the single issue resolution latency?

To explore this question, we develop the measurement of D_{Focus}. Then, we calculate the average D_{Focus} of developers in each issue as ***avgFocus***. Note that we still conduct the experiments on the issues from Rails project and TrinityCore project. D_{Focus} represents the uncertainty in developers' discussions focus switching behavior (or the diversity of focus switches) in a given week. The higher the D_{Focus}, the lower the entropy, the higher the developer's focus in this issue. On the contrary, the lower the D_{Focus}, the higher the entropy, the more scattered the developer's focus and developer has more multi-discussing behavior. Thus, ***avgFocus*** represents the average D_{Focus} of issues in a project.

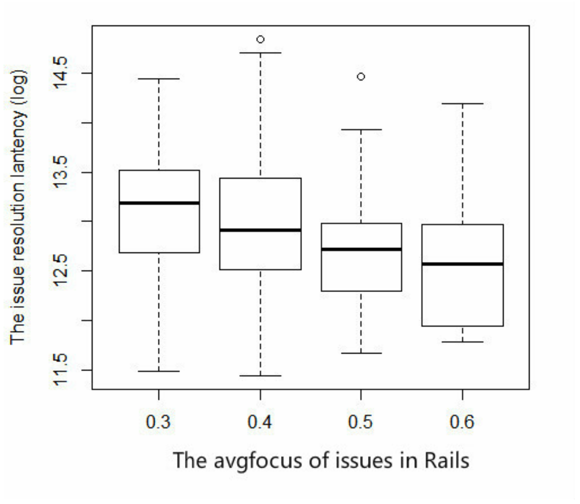


Figure 8: The single issue resolution latency vs ***avgFocus*** in Rails project.

Table V: Rails developer focus model. Resp.: log(singleIssueLatency), R^2=0.18.

	Estimate (Error)	Sum Sq.
(Intercept)	-0.3203 (0.1595)***	
avgFocus	-0.1318 (0.0431)***	11.85***
textLen	0.0021 (0.0425)	0.01
nDevelopers	0.5319 (0.0659)***	54.30***
nComments	-0.2441 (0.0662)***	13.29***
subMemberTag Ture	-0.3554 (0.2013).	2.17
hasLabelTag Ture	0.3639 (0.1664)*	3.95*

$***p < 0.001$, $**p < 0.01$, $*p < 0.05$, $.p < 0.01$

According to statistics, the value of ***avgFocus*** in over 98% issues of two projects is between 0.2 and 0.6. We mark the value of ***avgFocus*** from 0.2 to 0.3 as 0.3, from 0.3 to 0.4 as 0.4, from 0.4 to 0.5 as 0.5, and from 0.5 to 0.6 as 0.6. Figure 8 and Figure 9 show the boxplots of the relationship between the ***avgFocus*** of issue and the issue resolution in Rails project and TrinityCore project. We find that when ***avgFocus*** increases, the issue resolution latency in project decreases, *i.e.*, the less multi-discussing behavior in an issue, the more focus of developers on this single issue, and the shorter issue resolution latency. To better explore the effect of D_{Focus} on the single issue resolution of project to see how multi-discussing affects the single issue resolution, we build developer focus models. Table V shows the summary of the regression model (**Model-4**) in Rails project. The adjusted fit of this model is R^2=0.18.

The factor ***avgFocus*** is significant and has the most sizeable and negative effect on the issue resolution latency in Rails project (13.8% of the variance explained). E.g., 10% increase to ***avgFocus*** has the effect of decreasing the issue resolution lantency outcome by 1-$e^{-0.1318}$ =12.3%, holding all other variables constant. The factor ***nDevelopers*** has a significant positive effect on the single issue resolution latency (63.5% of the variance explained). 10% increase to ***nDevelopers*** has the effect of increasing the issue resolution latency outcome by 70.2%, holding all other variables constant. More participants may indicate that issue is a complex problem which need more developers to resolve. The factor ***nComments*** has a significant negative effect (15.5% of the variance explained). 10% increase to ***nComments*** has the effect of decreasing the issue resolution lantency outcome by 21.7%, holding all other variables constant. More comments may indicate that issue was focused by more developers, so the issue may be easily resolved. According to results, we find that with the increase of ***avgFocus***, the resolution time of the issue gradually decreases. It means that if the developer only focuses on discussing one issue, it will help improve the resolution of this issue.

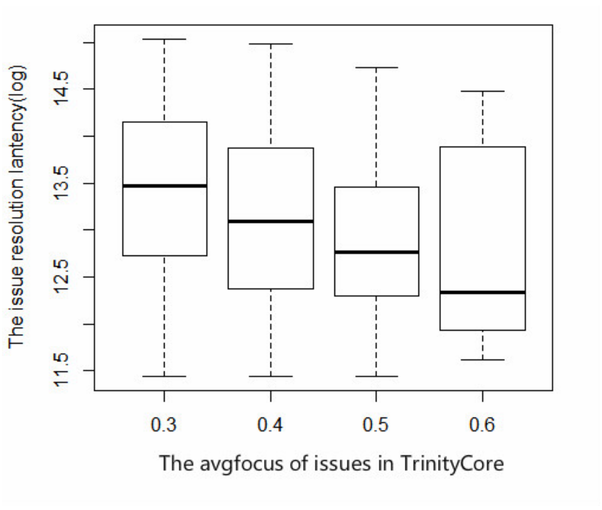


Figure 9: The single issue resolution latency vs ***avgFocus*** in TrinityCore project.

Table VI: TrinityCore developer focus model. Resp.: log(singleIssueLatency), R^2=0.19.

	Estimate (Error)	Sum Sq.
(Intercept)	-0.5695 (0.0987)***	
avgFocus	-0.1872 (0.0242)***	55.77***
textLen	-0.0036 (0.0242)	0.21
nDevelopers	0.3069 (0.0241)***	151.05***
nComments	0.0005 (0.0241)	0.03
subMemberTag Ture	0.5109 (0.0950)***	25.44***
hasLabelTag Ture	0.5687 (0.1020)***	25.57***

$***p < 0.001$, $**p < 0.01$, $*p < 0.05$

Table VI shows the summary of the model (**Model-5**) in TrinityCore project. The adjusted fit of this model is R^2=0.19. As expected, the factor ***avgFocus*** has a significant negative effect on the issue resolution latency (21.6% of the variance explained). E.g., 10% increase to ***avgFocus*** has the effect of decreasing the issue resolution latency outcome by 17.1%, holding all other variables constant. The factor ***nDevelopers*** has a significant positive effect on the single issue resolution latency (58.5% of the variance explained). 10% increase to ***nDevelopers*** has the effect of increasing the issue resolution latency outcome by 35.9%, holding all other variables constant. The factor ***nComments*** nearly has no effect in model. But the factor ***subMemberTag*** has a significant positive effect on the single issue resolution latency (9.8% of the variance explained). A one-unit increase in ***subMemberTag***, results in the increase in the issue resolution latency of 66.7%, holding all other variables constant. And the factor ***hasLabelTag*** has a significant positive effect on the single issue resolution latency (9.9% of the variance explained). A one-unit increase in ***hasLabelTag***, results in the increase in the issue resolution latency of 76.6%, holding all other variables constant. According to the results, we find that with the increase of ***avgFocus***, the resolution time of the issue gradually decreases. It means that if the developer only focuses on discussing this issue, it will help improve the resolution of the issue. Thus, we find that,

In a single issue, more discussion-switching means less focus of developers, which may bring longer issue resolution latency.

V. Discussion

Here, we distill some practice implications for researchers and developers. And we discuss the threats of our study.

A. Practice Implications

We have found that the phenomenon of developers multi-discussing in multiple issues is common in GitHub projects. Our study shows that multi-discussing has a positive effect on the overall issue resolution of a project, while it has a negative effect on the single issue resolution. Thus, we distill the following practice implications for researchers and developers.

(1) *The multi-discussing is good to the overall project development.*

Our result shows that multi-discussing between issues is common in developer community (RQ1) and tends to have a positive affect on the overall issue resolution of a project (RQ2). Switching discussion between issues in project allows developers to make more efficient use of their time and effort, and provides them with opportunities for learning and knowledge transfer. Hence, software teams could benefit from practices and tools that support issues assignment that account multi-discussing.

(2) *There are limitations on the current developers' multi-discussing.*

However, our result also shows that multi-discussing tends to have a negative affect on the single issue resolution (RQ3). We know that multi-discussing relies on developers' rationally allocating their time and focus, which may bring different influence to the resolution of different issues. In RQ3, we find that as the developer's ***avgFocus*** in discussions is dispersed, the resolution time of the single issue in project gradually increase. In other words, if developers should focus on this issue, but have too much multi-discussing behavior, it tends to have a negative affect on this issue resolution. Multi-discussing may improve the resolution efficiency of issues in whole project, but its impact on single issue could be limited. Thus, there may exist trade-offs when developers do the multi-discussing, considering the overall project issue resolution and the single issue resolution, which need to be further explored in the future.

B. Threats on Validity

Our dataset 631 projects. So it may not represent the universe of all real world projects. Furthermore, we have focused this study on open source projects found on GitHub which may not perfectly generalize to every software project. Nonetheless, to the best of our knowledge, GitHub is the largest database of projects and has no restriction on the types of project that can be hosted. But our results may not be appropriate for other commercial projects, which are not open source projects.

In our regression models, our factors may be considered incomplete. E.g., in the issue resolution latency model of rails project, we only considered the length of issue text, without analyzing their semantics. In future work, we will further analyze their semantics.

VI. Conclusion and Future Work

The phenomenon of developers multi-discussing across issues is common in open source ecosystems, like GitHub. This paper explores the relationship between multi-discussing and the issue resolution. We collect and analysis data from 631 GitHub open-source projects. Our results show that multi-discussing tends to have a positive affect on the overall issue resolution of a project, *i.e.*, more multi-discussing, shorter average issue resolution latency. However, multi-discussing seems to have a negative affect on the single issue resolution, *i.e.*, more discussion-switching may bring longer issue resolution latency. We also distill some implications for researchers and developers.

In the future, we will try to provide a more deeper investigation of the multi-discussing to advise developers how many issues being focused a week could bring the most beneficial to the development efficiency.

Acknowledgment

This work was supported by Laboratory of Software Engineering for Complex Systems as well as the National Grand R&D Plan (Grant No. 2018YFB1003903), and the National Natural Science Foundation of China (Grant No. 61502512 and 61432020).

References

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *International Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2012, pp. 1277–1286.

[2] R. F. Adler and R. Benbunan-Fich, "Juggling on a high wire: Multitasking effects on performance," *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 156–168, 2012.

[3] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov, "The sky is not the limit: multitasking across github projects," in *International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 994–1005.

[4] S. Aral, E. Brynjolfsson, and M. Van Alstyne, "Information, technology and information worker productivity: Task level evidence," 2007.

[5] V. M. González and G. Mark, "Managing currents of work: Multi-tasking among multiple collaborations," in *ECSCW*. Springer, 2005, pp. 143–162.

[6] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi, "On the influence of maintenance activity types on the issue resolution time," in *PMSE*. ACM, 2014, pp. 12–21.

[7] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 188–197.

[8] Y. Zhang, Y. Yu, H. Wang, B. Vasilescu, and V. Filkov, "Within-ecosystem issue linking: a large-scale study of rails," in *International Workship on Software Mining*. ACM, 2018, pp. 12–19.

[9] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: evaluating contributions through discussion in github," in *Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2014, pp. 144–154.

[10] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," in *International Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2000, pp. 319–328.

[11] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: Theory and measures," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 123–148, 2006.

[12] S. A. Barab, J. G. MaKinster, and R. Scheckler, "Designing system dualities," *Designing for virtual communities in the service of learning*, pp. 53–90, 2004.

[13] D. Hu, T. Wang, J. Chang, Y. Zhang, and G. Yin, "Bugs and features, do developers treat them differently?" in *ICAIBD*. IEEE, 2018, pp. 250–255.

[14] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in github," in *CSMR*. IEEE, 2013, pp. 323–326.

[15] Y. Wu, J. Kropczynski, P. C. Shih, and J. M. Carroll, "Exploring the ecosystem of software developers on github and other platforms," in *International Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2014, pp. 265–268.

[16] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "A exploratory study of @-mention in github's pull-requests," in *Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1. IEEE, 2014, pp. 343–350.

[17] Y. Zhang, H. Wang, G. Yin, T. Wang, and Y. Yu, "Exploring the use of @-mention to assist software development in github," in *Asia-Pacific Symposium on Internetware*. ACM, 2015, pp. 83–92.

[18] ——, "Social media in github: the role of @-mention in assisting software development," *Science China Information Sciences*, vol. 60, no. 3, p. 032102, 2017.

[19] G. Mark, V. M. Gonzalez, and J. Harris, "No task left behind?: examining the nature of fragmented work," in *SIGCHI conference on Human factors in computing systems (CHI)*. ACM, 2005, pp. 321–330.

[20] M. B. O'leary, M. Mortensen, and A. W. Woolley, "Multiple team membership: A theoretical model of its effects on productivity and learning for individuals and teams," *Academy of Management Review*, vol. 36, no. 3, pp. 461–478, 2011.

[21] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *IEEE working conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[22] Q. Xuan, A. Okano, P. Devanbu, and V. Filkov, "Focus-shifting patterns of oss developers and their congruence with call graphs," in *Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2014, pp. 401–412.

[23] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2008, pp. 308–318.

[24] G. Jin, W. Zhang, D. Deng, B. Liblit, and S. Lu, "Automated concurrency-bug fixing." in *OSDI*, vol. 12, no. 2012, 2012, pp. 221–236.

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

[26] A. Arcuri and X. Yao, "A novel co-evolutionary approach to automatic software bug fixing," in *CEC*. IEEE, 2008, pp. 162–168.