Information and Software Technology 115 (2019) 1–4

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Multi-reviewing pull-requests: An exploratory study on GitHub OSS projects

Dongyang Hu[*], Yang Zhang, Junsheng Chang, Gang Yin, Yue Yu, Tao Wang

Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, China

A R T I C L E   I N F O

Keywords:
Multi-reviewing
Pull-requests
GitHub

A B S T R A C T

Context: GitHub has enabled developers to easily contribute their review comments on multiple pull-requests and switch their review focus *between* different pull-requests, *i.e.*, multi-reviewing. Reviewing multiple pull-requests simultaneously may enhance work efficiency. However, multi-reviewing also relies on developers' rationally allocating their focus, which may bring a different influence to the resolution of pull-requests.
Objective: In this paper, we present an ongoing study of the impact of multi-reviewing on pull-request resolution in GitHub open source projects.
Method: We collected and analyzed 1,836,280 pull-requests from 760 GitHub projects to explore how multi-reviewing affects the resolution of a pull-request.
Results: We find that multi-reviewing is a common behavior in GitHub. However, more multi-reviewing behaviors tend to bring longer pull-request resolution latency.
Conclusion: Multi-reviewing is a complex behavior of developers, and has an important impact on the efficiency of pull-request resolution. Our study motivates the need for more research on multi-reviewing.

1. Introduction

By implementing the concept of *social coding*, GitHub creates a developer-friendly environment that enables developers to network, collaborate, and promote their projects [1]. Pull-request is the primary method for code contributions from thousands of developers. To maintain the quality of the contribution of pull-request, pull-request review is an essential part of distributed software development [2]. Such social coding contribution also leads to enhance pull-request resolution process, in which developers can easily participate in reviewing pull-requests. As a result, it is common to find that developers contribute their review comments on multiple pull-requests and switch their focus between pull-requests. In our study, we define it as *multi-reviewing*.

Fig. 1 gives an overview of the multi-reviewing process. During the resolution of pull-request A, many developers may participate in and comment, trying to review this pull-request. In particular, developer X may participate in A's review first (①). Then, although A was not closed, X switched his/her focus to another pull-request, B, and contributed a comment in B's review(②).

Next, although B was not closed, X moved to A's review again and contributed two comments (③ and ④). After that, X participated in B's review again (⑤). Thus, we find that X switched his/her focus between A and B over time.

Multi-reviewing, 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 [3]. Recently, many studies have explored the multitasking in GitHub Projects [4]. However, to the best of our knowledge, no existing research has been conducted on how developers do the multitasking between pull-requests, *i.e.*, at a pull-request level. To fill this research gap, we aim to explore how developers do the multi-reviewing and what the impact of multi-reviewing on pull-request resolution.

Multi-reviewing may enhance the work efficiency of developers, helping them review multiple pull-request simultaneously. However, multi-reviewing relies on developers' rationally allocating their time and focus, which may vary. Different strategies and focus of multi-reviewing may bring different effects to pull-request resolution. Exploring multi-reviewing behavior can help developers understand the benefits and limitations when they switch their review focus between pull-requests. Specifically, we ask:

[*] Corresponding author.
E-mail addresses: hudongyang17@163.com (D. Hu), yangzhang15@nudt.edu.cn (Y. Zhang).

https://doi.org/10.1016/j.infsof.2019.07.004
Received 26 February 2019; Received in revised form 10 June 2019; Accepted 11 July 2019
Available online 13 July 2019

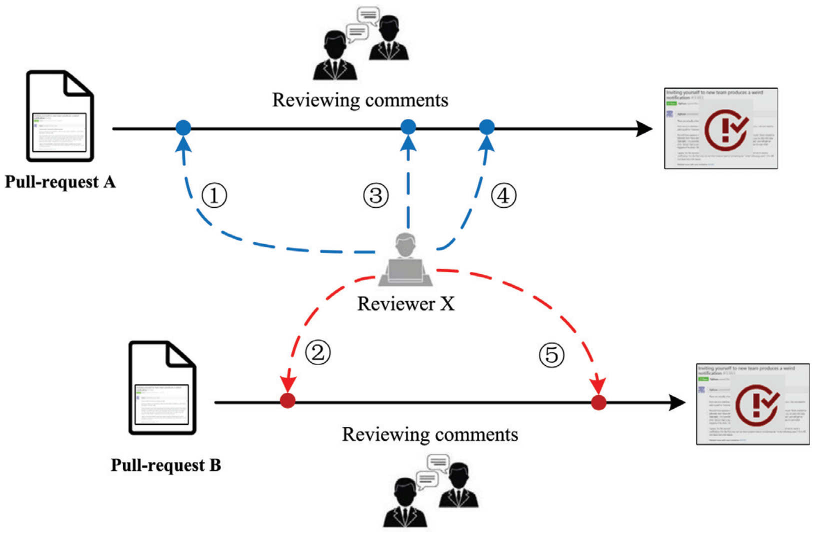

Fig. 1. Overview of the multi-reviewing process.

Table 1
Aggregate statistics of the 760 projects.

Statistic	Mean	St.Dev.	Min	Median	Max
#Members	12.6	30.3	2.0	6.0	336.0
#Forks	1159.8	2264.5	3.0	443.5	26,487.0
#Stars	4004.0	10,851.6	6.0	803.5	215,302.0
#Pull-requests	2247.0	845.1	1515.0	2180.0	4984.0

RQ1: *How many pull-requests in which the developers do the multi-reviewing?*
RQ2: *How does multi-reviewing affect the resolution of a pull-request?*

In summary, our findings are:

- On average, 62% of pull-requests in our dataset contains the multi-reviewing developers, but the percentage of different projects vary.
- More multi-reviewing may bring more focus switching, thereby causing longer pull-request resolution latency.

2. Methodology

2.1. Data set

We obtain the data from GHTorrent [5], which is an effort to create a scalable, offline mirror of data offered through the Github REST API. In our study, we select those large projects that have more than 1500 pull-requests as our studied projects. Finally, we collect 760 projects with 1,836,280 pull-requests. Table 1 shows the aggregate statistics.

2.2. Measuring the review-switching focus – $\mathrm{P}_{\mathrm{Focus}}$

To explore how multi-reviewing affects pull-request resolution, we develop the measurement P_{Focus}, to define the rate and breadth of developers' review-switching focus in a given week. In this work, we use the Shannon entropy index [4], to calculate the P_{Focus} as $\frac{1}{-\sum_{i=1}^{N} p_i \log_2 p_i + 1}$.

Considering a project developer Y in the *i*'th week, p_i is the fraction of the pull-request among all pull-requests that Y reviewed in this week, and N is the total number of pull-requests that Y reviewed during the *i*'th week. Thus, the higher the P_{Focus} value of the developer, the more focus on the review comments in a pull-request, the less multi-reviewing behaviors the developer performs in other pull-requests.

For example, Fig. 2 shows two examples of calculating the P_{Focus} between developer A and developer B. A and B review pull-requests both five times in a week. However, A reviewed three different pull-requests, *i.e.*, #33024, #14056, and #58210, and reviewed pull-request #33024 three times in a week. B reviewed four different pull-requests, *i.e.*, #33024, #21776, #14452, and #67024, and his reviews of pull-requests were relatively scattered in a week. So the P_{Focus} (the value is 0.73) of A is higher than the P_{Focus} (the value is 0.52) of B.

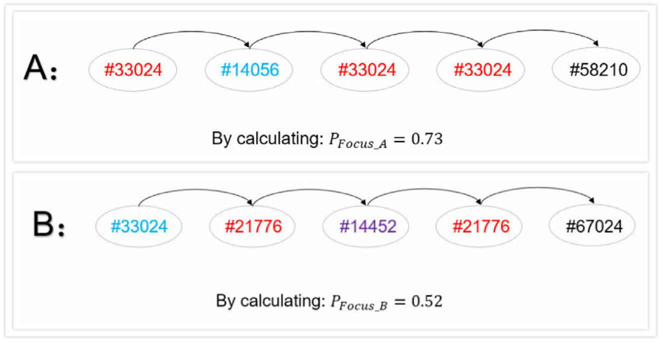
A:
#33024
#14056
#33024
#33024
#58210
By calculating: $P_{Focus\_A} = 0.73$
B:
#33024
#21776
#14452
#21776
#67024
By calculating: $P_{Focus\_B} = 0.52$

Fig. 2. Examples of calculating the P_{Focus}.

2.3. Regression analysis

To study how multi-reviewing affects pull-request resolution, we build the mixed-effects linear regression model (using packages `lme4` and `lmerTest` in R), *i.e.*, *review focus model*, with the random-effect factor *programming language* of the project. All other variables are modeled as fixed effects. The dependent variable (outcome) of this model is ***PRLatency***. It is the resolution latency of a pull-request, in minutes, as a proxy for a pull-request resolution speed. In our study, we only consider a pull-request latency between a pull-request creation to its first closing date.

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

- ***avgFocus***: average P_{Focus} value of all developers in the pull-request, as a proxy for the developers' review focus.
- ***nComments***: total number of comments in this pull-request, as a proxy for the review length.
- ***nParticipants***: total number of developers that participated in this pull-request, as a proxy for the work effort of this pull-request.
- ***textLen***: total words length of a pull-request text (title and description), as a proxy for pull-request importance and complexity. Longer

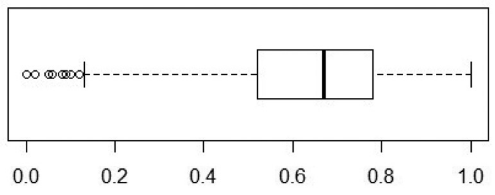

Fig. 3. Distribution of the percentage of multi-reviewed pull-requests.

descriptions may indicate higher importance and complexity of pull-request or better documentation [6].

- ***nPRs***: total number of pull-requests 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 [7].
- ***nStars***: total number of stars in the project, as a proxy for the popularity of the project [8].

In our models, the variance inflation factors, which measure multicollinearity [9] 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 the sum of squares (via ANOVA). We consider 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 [10].

3. Study results

3.1. RQ1: How many pull-requests in which the developers do the multi-reviewing?

To answer the question, we calculate the percentage of pull-requests that contains at least one multi-reviewing developer in the 760 projects. During one pull-request lifetime, *i.e.*, from its creation to its first closing date, if at least one of its developers also reviewed other pull-request(s), we call this pull-request as *multi-reviewed pull-request*. Fig. 3 shows the boxplot of the percentage of multi-reviewed pull-requests among the 760 projects. On average, the percentage of multi-reviewed pull-requests is 62.4% (median of 67.0%). However, we find that the percentage of multi-reviewed pull-requests in different projects vary. Thus, we find that,

On average, in 62% of pull-requests of our dataset, the participants also multi-reviewing other pull-request(s).

3.2. RQ2: How does multi-reviewing affect the resolution of a pull-request?

During a pull-request resolution, there may exist one or many developers to review. Some of them may perform the multi-reviewing, some of them may not, which may bring different effects to pull-request resolution latency. We calculate the average P_{Focus} of developers in each pull-request as ***avgFocus***. *AvgFocus* represents the uncertainty in developers' focus switching behaviors (or the diversity of focus switches) in a given week. The higher the *avgFocus*, the lower the entropy, the higher the developer's focus in this pull-request. On the contrary, the lower the *avgFocus*, the higher the entropy, the more scattered the developer's focus and developer has more multi-reviewing behaviors.

The value of *avgFocus* in over 98% pull-requests of our dataset is between 0 to 16 approximately. We mark the value of *avgFocus* from 0 to 2 as 2, from 2 to 4 as 4, and so on. Fig. 4 shows the boxplots of the relationship between the *avgFocus* of pull-request and the resolution latency. We find that when *avgFocus* increases, pull-request resolution latency decreases, *i.e.*, the less multi-reviewing behaviors in a pull-request, the more focus of developers on this pull-request, and the shorter resolution latency.

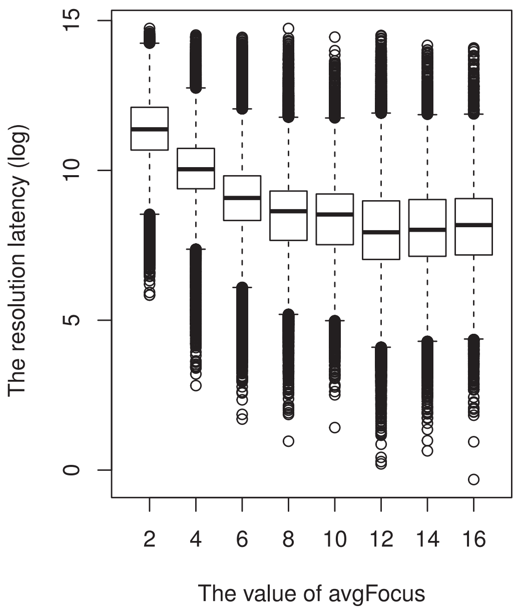

Fig. 4. Pull-request resolution latency vs *avgFocus*.

Table 2
Review focus model. Resp.: log(PRLatency), $R^2_m = 0.24$, $R^2_c = 0.31$.

	Estimate (Error)	Sum Sq.
(Intercept)	0.0563 (0.0407)	
avgFocus	−0.4543 (0.0019)***	41,276***
textLen	0.0429 (0.0019)***	377***
nParticipants	0.0402 (0.0021)***	254***
nComments	0.1200 (0.0021)***	2,287***
nPRs	−0.0352 (0.0020)***	223***
nMembers	0.0202 (0.0020)***	74***
nForks	0.0909 (0.0026)***	877***
nStars	−0.0134 (0.0026)***	19***

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

Further, Table 2 shows the summary of *review focus model*. In total, the model explained $R^2_c = 0.31$ of the deviance ($R^2_m = 0.24$). The factor *avgFocus* is significant and has the most sizeable and negative effect on pull-request resolution latency (90.9% of the variance explained). Holding all other variables constant, for a one-unit increase in avgFocus, the expected decrease in PRLatency is 36.5% ($1-e^{-0.4543}$). Thus, we find that,

In a pull-request, more review-switching means less focus of participants, which may bring longer pull-request resolution latency. Thus, more multi-reviewing behaviors within a pull-request may retard the resolution process.

4. Research agenda

Our study provides a rich resource of initial ideas for further study.

Further analyze the barriers and benefits when multi-reviewing. Our study provides a preliminary study of the multi-reviewing in GitHub, including its basic usage and effect on pull-request resolution. Multi-reviewing may improve the developers' work efficiency, but it may also disperse the developers' focus. Hence, future work should further investigate the barriers and benefits of the multi-reviewing in pull-request mechanism.

Explore the effect of the particular number of pull-requests on multi-reviewing outcomes. We found that different focus may affect pull-request resolution. Developers may have a choice of the number of pull-requests reviewed, and that reviewing different number of pull-requests associates with different outcomes. Therefore, developers should pay attention to the review workload, and researchers should investigate the barriers and benefits developers face when reviewing a particular number of pull-requests. Moreover, the issue of how to manage and help developers review the appropriate number of pull-requests needs to be addressed.

Investigate the differences of participants between more dimensions during multi-reviewing. Our quantitative study mostly focused on comparing the review focus between different developers. How the multi-reviewing participants differ in other dimensions should be further empirically evaluated. E.g., it would be interesting to study participants' gender differences in multi-reviewing, since researches have demonstrated that gender in social science is also a significant factor in software development. With much more data and careful developers classification along different dimensions, some patterns may become apparent.

Design automatic tools to support the multi-reviewing. We find that multi-reviewing has a significant impact on developers collaboration. However, there is still a lack of automatic tools that support the multi-reviewing. Tool builders should design and develop some automatic tools, *e.g.*, a pull-request prioritization tool to prioritize pull requests that developers need to review, helping developers focus on those important tasks.

5. Conclusion and future work

This paper explores the relationship between multi-reviewing and pull-request resolution. We collected and analyzed 1,836,280 pull-requests from 760 GitHub open-source projects. Our results show that more review-switching may bring longer resolution latency of a pull-request.

In future work, we plan to conduct an in-depth analysis of the experiences and expectations when developers multi-reviewing pull-requests. Furthermore, we plan to optimize our regression models and develop automatic tools to guide developers' multi-reviewing operation.

Declarations of interest

None.

Acknowledgments

This work was supported by Laboratory of Software Engineering for Complex Systems as well as the National Grand R&D Plan (Grant no. 2018YFB1004202), and the National Natural Science Foundation of China (Grant no. 61702534).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.infsof.2019.07.004.

References

[1] 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.
[2] Y. Yu, H. Wang, G. Yin, T. Wang, Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? Inf. Softw. Technol. 74 (2016) 204–218.
[3] R.F. Adler, R. Benbunan-Fich, Juggling on a high wire: multitasking effects on performance, Int. J. Hum.-Comput. Stud. 70 (2) (2012) 156–168.
[4] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, V. Filkov, The sky is not the limit: multitasking across GitHub projects, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, 2016, pp. 994–1005.
[5] G. Gousios, D. Spinellis, Ghtorrent: GitHub's data from a firehose, in: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, 2012, pp. 12–21.
[6] Y. Yu, H. Wang, V. Filkov, P. Devanbu, B. Vasilescu, Wait for it: determinants of pull request evaluation latency on GitHub, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, 2015, pp. 367–371.
[7] A.G. Silvius, R. Schipper, A conceptual model for exploring the relationship between sustainability and project success, Procedia Comput. Sci. 64 (2015) 334–342.
[8] K. Aggarwal, A. Hindle, E. Stroulia, Co-evolution of project documentation and popularity within GitHub, in: 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 360–363.
[9] R. York, Residualization is not the answer: rethinking how to address multicollinearity, Social science research 41 (6) (2012) 1379–1386.
[10] A. Arcuri, X. Yao, A novel co-evolutionary approach to automatic software bug fixing, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 162–168.