Review Dynamics in Open-Source Software: A Case Study of OpenStack

Siqi Liu
University of Waterloo
Waterloo, Ontario, Canada
sq2liu@uwaterloo.ca

***Abstract*—Code review is an integral part of the modern software engineering practice. In open-source software, code review is done in a collaborative setting where all review activities are visible to the public. In this paper, we perform a case study on OpenStack, a large-scale open-source cloud operating ecosystem, to understand how visible information may influence the evaluation decision of a reviewer and the software quality. We select three core projects within OpenStack and analyze their code review dynamics. We then build statistical models to determine whether differences in review dynamics are associated with differences in software qualities. Our findings suggest that reviewers are more likely to provide a positive vote when there are existing positive votes. However, we find that such reviewer dynamics have minor impact in terms of software quality.**

***Index Terms*—code review, software quality, open-source software**

I. Introduction

One of the main goals of conducting peer code review is to find and eliminate defects from a patch [2]. Thongtanunam and Hassan [1] examined the code review dynamics in OpenStack and Qt as well as their impacts on software quality. In this paper, we attempt to replicate their results using only OpenStack. We use the same OpenStack projects as Thongtanunam and Hassan did in their original paper to fit our models, and we select another OpenStack project as validation to our models. We find that review dynamics, particularly the proportion of prior votes that are positive, have a significant impact on the evaluation decision of the reviewer. We also find that these review dynamics have a relatively small impact on software quality. Replication package of our work is available online.[1]

A. Paper Organization

Section II provides a background for our work. Section III describes the design of our case study. Section IV and V present the results of our analyses. Section VI discusses our results and compares them with the results found in Thongtanunam and Hassan [1]. Section VII discusses possible threats. Finally, Section VIII draws conclusions.

II. Background

In this section, we provide an overview of OpenStack, its code review processes, and the work done by Thongtanunam and Hassan [1].

A. OpenStack Overview

OpenStack is a free, open standard cloud operating system. Since its inception in 2010 as a joint project of Rackspace Hosting and NASA, OpenStack has grown substantially and is now being actively maintained by more than 500 companies and thousands of community volunteers. OpenStack is broken up into service areas and projects. At the time this report was written, there are 29 individual projects within OpenStack.[2] Even though each project is managed and developed independently, they share common development infrastructures [5].

B. OpenStack Code Review Process

The development workflow at OpenStack is centred around Gerrit, which uses the idea of patches rather than pull requests. To propose changes to a project, an author starts by cloning the master branch of the project repository. After changes have been made locally, the author can propose the changes to Gerrit using the *git-review* tool, creating a patch. The proposed changes are then picked up by the Continuous Integration (CI) tool, which runs *check tests* on the changes, while human reviewers review and vote on the patch. Depending on the check test results and the feedback from the reviewers, the patch author may need to amend the changes and propose additional patch sets. Once the patch has been approved by both the CI tool and the reviewers, the CI tool runs *gate tests* to detect any merge conflict before finally merging it into the master branch.

C. Prior Work

Thongtanunam and Hassan [1] presented the first case study on the review dynamics and its effects on the software quality. Through their study, they found that

- The proportion of prior positive votes and prior reviewer comments are highly associated with the evaluation decision of a reviewer
- The co-working frequency of the reviewer with the patch author also has a positive relationship with the likelihood of providing a positive vote
- The review dynamic metrics have a relatively small relationship with the software quality

[1] https://github.com/david-siqi-liu/review-dynamics-openstack

[2] https://www.openstack.org/software/project-navigator/openstack-components

In this paper, we attempt to replicate their results by conducting the same set of analyses. However, we only analyze OpenStack, as prior study has shown that different organizational structures could have an impact on the review dynamics [3]. We extend prior work by introducing a new project from OpenStack that we will use for model validation. This allows us to understand the generalizability of our models.

III. Case Study Design

In this section, we provide an overview of our study and the studied projects.

A. Study Overview

We perform a two-fold analysis to better understand the review dynamics and their impact on the software quality.

1) Evaluation Decision: For each studied project, we mine its historical patches and patch characteristics from its Git repository. For each patch, we collect review dynamic variables based on the review activities on Gerrit. We also extract reviewer characteristics based on his/her past activities. We then build a linear mixed-effects logistic model to model the likelihood of receiving a positive vote. After validating our model against the validation project, we examine the relationship between our variables and the evaluation decision.

2) Software Quality: Based on findings from our first analysis, we form hypotheses and formulate variables that we use for our second analysis. We build a generalized linear model to predict the defect proneness of each patch. Once again, we examine the relationship between our proposed metrics and the software quality after validating our model against the validation project.

B. Studied Projects

We use the same set of projects as Thongtanunam and Hassan [1] did in their study for model fitting. In addition, we select another project from OpenStack for model validation. The project we select is Sahara, which is a data processing framework. We select Sahara as our validation project because it comes from a different service area, is being actively maintained, and shares similar target ratios for both analyses.

TABLE I
Studied Projects

	Glance	**Cinder**	**Neutron**	**Sahara**
Set	Train	Train	Train	Validation
# Patches	2,936	8,518	10,575	3,164
# Reviewers	626	1,246	1,332	245
Avg # Reviewers per Patch	4	4	5	4
% Patches w/ >1 Reviewers	94%	96%	100%	88%
% Reviews w/ Positive Votes	92%	88%	90%	94%
% Patches Fixing Inducing	57%	52%	60%	50%

Table I shows some high-level statistics of the projects we use in our study. We only use patches submitted from November 2011 to July 2019. This is to eliminate uncertainties from early adoption and to allow enough time for the patches to be reviewed and tested for defects after they have been merged. Nearly all of the patches have multiple reviewers, and the average number of reviewers is quite consistent throughout the projects. Around 90% of the reviews end up with a positive vote, which is expected since patch authors often make multiple amendments with the help of reviewers to get their proposed changes accepted. Lastly, more than half of the patches end up causing defects in the future.

IV. Analysis 1: Evaluation Decision

In this section, we present the data pipeline, the model and the results of our first analysis, which focuses on the evaluation decision of a review.

A. Data Extraction

We use PyDriller[3] to mine patches and extract their characteristics from Git repositories. For each patch, we collect its review data using the Gerrit REST API.[4] Lastly, we compute reviewer characteristics and assign them to each reviews. We assign the variables into three data dimensions - patch, review dynamics, and reviewer. Detailed list of variables and their descriptions are included in our supplementary material.[5]

B. Data Cleaning

We conduct correlation analysis to remove variables that are highly correlated. The metric we use is Spearman's rank correlation coefficient (ρ). For each pair of variables with an absolute correlation coefficient ($|\rho|$) greater than 0.7, we remove one of the variables from our study. For consistency, we use the `AutoSpearman` function in R.

C. Model Fitting

We use the linear mixed-effects model as our base model type. Linear mixed-effect model is an extension to simple linear model to account for both fixed and random effects. This type of model is used when there is a hierarchical structure to the data, in which case the variability in the outcome may come from either within the group or between the groups. In our case, the random-effects/groups are the reviewers. For example, some reviewers may be more lenient and tend to give more positive votes than other reviewers.

Since our goal is to predict whether a vote is positive or not, we use `family='binomial'` in our model configuration. To add the random effect, we include `(1|ReviewerID)` in our model formulas.

We fit five models to explore the explanatory power of each of the three data dimensions. The full model includes variables from all three data dimensions and the random-effect variable. Three models, each excluding one of the data dimensions, are also fitted. Lastly, we fit a null model which only contains the random-effect variable.

[3] https://pydriller.readthedocs.io/en/1.8/

[4] https://review.opendev.org/Documentation/rest-api.html

[5] https://zenodo.org/record/4695052

D. Model Evaluation

We use Area Under the ROC Curve (AUC) scores to evaluate model performance. Higher AUC score represents better discriminatory power. As shown in Table II, both the *Ex-Reviewer* model and the *Full* model are able to achieve an AUC score that is 14% higher than the *Null* model. This implies that even if we exclude all of the reviewer variables, our model performance wouldn't deteriorate as much.

TABLE II
ANALYSIS 1 - AUC SCORES

Model	AUC	× of Null AUC
Null	0.72	
Ex-Patch	0.81	1.12
Ex-Dynamic	0.77	1.06
Ex-Reviewer	0.82	1.14
Full	0.82	1.14

We conduct likelihood-ratio (LR) tests to evaluate the explanatory power of each of the data dimensions. The results, shown in Table III, show that while all three data dimensions are statistically significant, review dynamic variables offer significantly more explanatory ability than variables in the other two data dimensions.

TABLE III
ANALYSIS 1 - LR TEST RESULTS

Model A	Model B	Δ D.F.	LR_{χ^2}	% of Full LR_{χ^2}
Null	Full	21	8,768	
Ex-Patch	Full	11	1,512	17%
Ex-Dynamic	Full	5	5,934	68%
Ex-Reviewer	Full	5	326	4%

Based on results from the two tests above, we select the *Ex-Reviewer* model to be our final model since reviewer variables offer little explanatory power and AUC score boost. We calculate the Wald statistics for each variable in our final model. The larger the Wald statistics a variable possesses, the stronger its relationship with the target variable. Table IV shows the Wald statistic, p-value significance level, and the sign for each variable.

E. Model Validation

To examine the generalizability of our model, we test it against the validation project. Since we used a mixed-effect model with reviewers being the groups, our model has not seen the reviewers in the validation project and only the fixed-effect variables can be used for prediction. We achieve an AUC score of 0.73 on the validation project, as opposed to the 0.82 AUC score we see on the training set. This implies that there is enough variability among the reviewers that we are unable to form accurate predictions by using the fixed-effect variables alone.

F. Findings

Review dynamics have a stronger relationship with the evaluation decision than patch and reviewer characteristics. Even though the review dynamics data dimension only has 5 variables, it captures 68% of the total LR. Moreover, *% Prior Votes Positive* is the most statistically significant variable overall. Its positive sign indicates that it is more likely for a reviewer to provide a positive vote when there are more positive votes among prior votes.

TABLE IV
ANALYSIS 1 - WALD STATISTICS

	Wald_{χ^2}	Significance	Sign
Intercept	1,656	***	+
Dimension: Patch			
# Lines Added	547	***	-
# Lines Deleted	2,109	***	+
# Lines of Code	†		
# Files Impacted	†		
# Directories Impacted	86	***	-
Average Cyclomatic Complexity	10	**	-
Entropy	460	***	-
Is Bug Fixing	6	*	+
Description Length	239	***	-
# Prior Commits	†		
# Prior Commits Bug Fixing	†		
Average Prior Commits Age	8	**	-
Is Author Core Developer	102	***	+
Is Author Experienced Author	26	***	+
Is Author Experienced Reviewer	51	***	+
Dimension: Review Dynamics			
# Prior Votes	39	***	-
# Prior Comments	5	*	-
% Prior Votes Positive	4,013	***	+
% Prior Positive Votes From Core Developers	23	***	+
% Prior Negative Votes From Core Developers	33	***	-

Statistical significance: *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, $\circ$ $p \geq 0.05$
† Removed in correlation analysis

Reviewers are more likely to provide a positive vote when the patch is not complex and simplifies the code base. This is evident in the signs in *# Lines Added* (-), *# Lines Deleted* (+), and *Entropy* (-). In addition, patch author's experience level and core developer status also have a statistically significant positive relationship with the likelihood of receiving a positive vote.

V. ANALYSIS 2: SOFTWARE QUALITY

In this section, we present the social variables, the model and the results of our second analysis, which focuses on the software quality of the merged patches.

A. Social Variables

Based on our findings in Analysis 1, we formulate 5 social variables for measuring review dynamics. These social metrics, shown in Table V, along with the patch variables from Analysis 1 and a couple of summary review variables[5] are used to model the software quality. Note that in this analysis, we will only be studying the merged patches.

B. Data Cleaning

We conduct correlation analysis to remove variables that are highly correlated, using the same method and threshold score as we do in Analysis 1.

TABLE V
ANALYSIS 2 - SOCIAL VARIABLES

Finding from Analysis 1	Variable	Description
Higher % prior votes positive ⟶ More likely to provide positive vote	% Positive Voters Consistent w/ Prior Votes Positive	Percentage of reviewers who provided a positive vote when at least one prior vote is positive
Higher % prior positive votes from core developers ⟶ More likely to provide positive vote	% Positive Voters Consistent w/ Prior Core Positive Votes	Percentage of reviewers who provided a positive vote when at least one prior positive vote comes from a core developer
Higher % prior negative votes from core developers ⟶ Less likely to provide positive vote	% Positive Voters Inconsistent w/ Prior Core Negative Votes	Percentage of reviewers who provided a positive vote when at least one prior negative vote comes from a core developer
Higher # prior votes ⟶ Less likely to provide positive vote	Average # Prior Votes for Positive Voters	Average number of prior votes for all reviewers who voted positive
Higher # prior comments ⟶ Less likely to provide positive vote	Average # Prior Comments for Positive Voters	Average number of prior comments for all reviewers who voted positive

C. Model Fitting

Unlike in Analysis 1 where we controlled for the variability within the reviewers, here we use the generalized linear model as our base model type. A patch is fix-inducing if there is a future patch that is bug fixing and modifies the same line. Our goal is to predict whether a merged patch is fix-inducing or not. Once again, we fit five models to explore the explanatory power of the three data dimensions.

D. Model Evaluation

Table VI and Table VII show the AUC scores and LR test results for the models. We select the *Ex-Review* model to be our final model since the summary review variables offer very little impact. Lastly, the Wald statistics for the final model variables are shown in Table VIII.

TABLE VI
ANALYSIS 2 - AUC SCORES

Model	AUC	× of Null AUC
Null	0.50	
Ex-Patch	0.67	1.33
Ex-Review	0.78	1.57
Ex-Social	0.78	1.55
Full	0.78	1.57

TABLE VII
ANALYSIS 2 - LR TEST RESULTS

Model A	Model B	Δ D.F.	LR_{χ^2}	% of Full LR_{χ^2}
Null	Full	17	5,402	
Ex-Patch	Full	12	3,373	62%
Ex-Review	Full	2	98	2%
Ex-Social	Full	3	359	7%

E. Model Validation

Once again, we test our model against the validation project. We are able to achieve an AUC score of 0.73 on the validation project, 6% below the AUC score on the training set (0.78). We believe this indicates that our results can be generalized to other projects within the same software ecosystem that share common development infrastructure.

TABLE VIII
ANALYSIS 2 - WALD STATISTICS

	Wald_{χ^2}	Significance	Sign
Intercept	402	***	+
Dimension: Patch			
# Lines Added	399	***	+
# Lines Deleted	34	***	-
# Lines of Code	†		
# Files Impacted	†		
# Directories Impacted	77	***	+
Average Cyclomatic Complexity	10	**	+
Entropy	438	***	+
Is Bug Fixing	127	***	+
Description Length	84	***	+
# Prior Commits	†		
# Prior Commits Bug Fixing	94	***	+
Average Prior Commits Age	496	***	-
Is Author Core Developer	144	***	-
Is Author Experienced Author	0	○	-
Is Author Experienced Reviewer	1	○	-
Dimension: Social			
% Positive Voters Consistent w/ Prior Votes Positive	212	***	+
% Positive Voters Consistent w/ Prior Core Positive Votes	238	***	-
% Positive Voters Inconsistent w/ Prior Core Negative Votes	1	○	-
Average # Prior Votes for Positive Voters	†		
Average # Prior Comments for Positive Voters	†		

Statistical significance: *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, ○ $p \geq 0.05$
† Removed in correlation analysis

F. Findings

Unlike in Analysis 1 where the review dynamics variables have a strong relationship with the target variable, here the relationship is not as strong as the patch characteristics. *Average Prior Commits Age* is the most significant variable. Its positive sign indicates that a patch is more prone to defects when it modifies lines that haven't been modified in a long time. Other variables, such as *# Lines Added* and *Entropy*, suggest that a patch is more prone to defects when its modifications are more complex and dispersed. Lastly, we find that whether the author is experienced or not is not statistically significantly associated with the patch quality.

Among the social variables, we see that when more reviewers follow prior positive votes, the approved patch is more prone to defects. However, if the prior positive votes come from core developers, then the approved patch is actually less prone to defects.

VI. Discussion

We now discuss the implications of our results and compare them with the findings in Thongtanunam and Hassan [1].

A. Review Dynamics

Our results in Analysis 1 show a significant relationship between the review dynamics and the evaluation decision of a reviewer. Most notably, reviewers are more likely to cast a positive vote when there are prior votes that are also positive. However, correlation does not mean causation, and a controlled experiment needs to be done to see if hiding prior votes leads to a change in voting outcomes.

On the other hand, the social variables we propose based on these findings have minor impact on the software quality, as shown in Analysis 2. Considering other expectations in a code review [2], [7], we believe that it is still beneficial for code reviews to be collaborative and open.

B. Patch Characteristics

Our analyses show that reviewers tend to favour patches that are less complex (Analysis 1), possibly because they are less prone to errors (Analysis 2). Even though patch characteristics are not as strongly correlated with the evaluation decision as the review dynamics are, patch characteristics have the majority of explanatory power when it comes to the software quality. This suggests that practitioners should propose simpler patches in order to avoid defects and promote acceptance.

C. Reviewer Characteristics

Reviewer characteristics, including interaction frequency with the patch author, do not have a significant impact on the evaluation decision. However, the variability within the reviewers make our mixed-effect model perform poorly on the validation project. Further research needs to be done to understand more about this variability.

D. Comparison With Prior Work

Overall, our results align with the results in the prior study by Thongtanunam and Hassan [1]. Both studies show a significant relationship between the review dynamics and the evaluation decision, and a minor relationship with the software quality. However, we note several key differences:

1) Our analysis does not show a significant relationship between the reviewer's interaction frequency with the patch author and the evaluation decision
2) Our analysis shows a significant negative relationship between the complexity of the patch and the likelihood of receiving a positive vote, while the relationship in the prior study is not as strong
3) Our analysis does not show a significant relationship between the summary review variables and the software quality (2% of full LR_{χ^2}), while the relationship in the prior study is much stronger (15%)

We believe that these key differences are due to discrepancies in the underlying data. Although Thongtanunam and Hassan provided replication datasets[6] for their case study, we do not have the scripts that generated these datasets. As such, we extract our data independently. After conducting thorough spot-checks and reconciliation between the replication dataset and the dataset we generate, we identify some gaps in the data:

1) Automated votes by CI tools are included in the replication dataset. This accounts for roughly 5% of all votes
2) The replication dataset misclassifies vote removals as negative votes
3) Over 70% of reviews in the replication dataset have no prior vote, implying that these patches only have one reviewer, which is not the case

We also encounter issues with PyDriller regarding methods that apply the SZZ algorithm to compute prior commits.[7] These issues directly affect several key variables in our study. Since we do not know which version of PyDriller the authors used, we are unable to fully replicate their dataset.

VII. Threats to Validity

We identify several new threats to validity in addition to the ones identified in Thongtanunam and Hassan [1].

External Validity. The purpose of the validation project is to examine the generalizability of our models. In Analysis 1, we show that our model does not perform well on the validation project, and we believe that this is due to the variability within the reviewers. However, it is also possible that the sub-optimal result is due to organizational differences between the validation project and the training projects.

Internal Validity. We are unable to fully reconcile the dataset we generate with the replication dataset due to reasons listed in Section VI-D. Nonetheless, our studies show similar findings and additional studies on other open-source software could help to corroborate our findings.

VIII. Conclusion

Code review in open source software is done in a collaborative and transparent setting. However, based on our case study, we find that such visible information may have an impact on the evaluation decisions of the reviewers. We show empirical evidence that review dynamics, particularly the proportion of prior votes that are positive, are highly correlated with the evaluation decision. On the other hand, these review dynamics have minor impact on the software quality. Our results on patch characteristics suggest that developers should propose less complex patches to improve software quality and likelihood of receiving positive votes. Furthermore, our study shows that due to the variability within the reviewers, our model for evaluation decision may not be generalizable to other projects, even within the same ecosystem. Nevertheless, we believe that our findings would be still of value to practitioners looking to improve their code review processes.

[6] https://zenodo.org/record/3556933

[7] https://github.com/ishepard/pydriller/issues/155

References

[1] Thongtanunam, Patanamon, and Ahmed E. Hassan. “Review dynamics and their impact on software quality.” IEEE Transactions on Software Engineering (2020).

[2] Bacchelli, Alberto, and Christian Bird. “Expectations, outcomes, and challenges of modern code review.” 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013.

[3] Walton, Christopher. “The Open Source Software Ecosystem.” 2004.

[4] Śliwerski, Jacek, Thomas Zimmermann, and Andreas Zeller. “When do changes induce fixes?.” ACM sigsoft software engineering notes 30.4 (2005): 1-5.

[5] German, Daniel M., et al. ““ Was My Contribution Fairly Reviewed?” A Framework to Study the Perception of Fairness in Modern Code Reviews.” 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018.

[6] Hassan, Ahmed E. “Predicting faults using the complexity of code changes.” 2009 IEEE 31st international conference on software engineering. IEEE, 2009.

[7] Sadowski, Caitlin, et al. ”Modern code review: a case study at google.” Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. 2018.