Impact of Continuous Integration on Code Reviews

Mohammad Masudur Rahman Chanchal K. Roy
Department of Computer Science, University of Saskatchewan, Canada
{masud.rahman, chanchal.roy}@usask.ca

***Abstract*—Peer code review and continuous integration often interleave with each other in the modern software quality management. Although several studies investigate how non-technical factors (e.g., reviewer workload), developer participation and even patch size affect the code review process, the impact of continuous integration on code reviews is not yet properly understood. In this paper, we report an exploratory study using 578K automated build entries where we investigate the impact of automated builds on the code reviews. Our investigation suggests that successfully passed builds are more likely to encourage new code review participation in a pull request. Frequently built projects are found to be maintaining a steady level of reviewing activities over the years, which was quite missing from the rarely built projects. Experiments with 26,516 automated build entries reported that our proposed model can identify 64% of the builds that triggered new code reviews later.**

I. Introduction

Quality assurance is one of the most important steps of software change management which is often done using a combination of manual (i.e., code reviews) and automated processes (i.e., continuous integration). GitHub, one of the most popular software ecosystems, offers a set of features for software quality management through pull requests (i.e., for code reviews) and automatic build supports (e.g., collaboration with Travis-CI). While peer code reviews involve manual checking of coding standard violations and simple logical errors in a submitted patch by the developers, continuous integration prevents the regression of a system by ensuring that the patch passes all the unit tests and merges with the system successfully. In the pull-based modern software development, automatic builds are often interleaved with or followed by code reviews by the developers [3]. Fig. 1 shows a growing trend for adopting automated software builds and code reviews by 1000+ open source projects from GitHub over the last six years. Peer code reviews are reported to be effective for improving coding standards [9], design quality [8] and overall quality [7] of a software system. There have been also several studies on how non-technical factors [2], developer participation [5] and even patch size [4, 10] affect the code reviews. Unfortunately, the impact of continuous integration on code review process is not yet properly understood given that they are interleaving steps in the software quality management. In particular, how automated software builds and tests might influence the participation or overall quality of code reviews is not substantially studied.

In this paper, we report an exploratory study where we analyze the recorded logs of the thousands of automated builds performed on the open source projects at GitHub, and find out how they might affect the peer code review activities on the same projects. In particular, we investigate whether the status and frequency of the automated builds correlate to the participation or quality of the code reviews. Such findings are likely to help us better understand how the manual code review process could be complemented with automatic tool supports. We thus answer three research questions as follows:

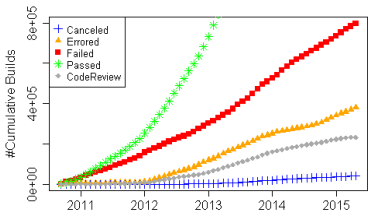

Fig. 1. Automated software build status and code reviews over the years

- **RQ$_1$:** Does the status of automated builds influence the code review participation in open source projects?
- **RQ$_2$:** Do frequent automated builds help improve the overall quality of peer code reviews?
- **RQ$_3$:** Can we automatically predict whether an automated build would trigger new code reviews or not?

Exploratory study using 578K automated build entries from 1000+ open source projects suggested important correlations between continuous integration and code review activities. First, automated build status has a notable impact on code review participation in the projects. Passed builds are more likely to trigger new code reviews for a pull request than the other builds. Second, automated build frequency has a major role in improving code reviews of the open source projects. Our investigation suggests that frequently built projects are likely to maintain a steady level of reviewing activities over the years, which was quite missing from the rarely built projects. Experiment using 26,516 build entries reported that our model can identify 64% of the builds that triggered new code reviews later, which is promising. Our model can offer automatic supports in the code reviews and software quality management by identifying the appropriate pull requests for code reviews.

II. Data Collection

We collect a total of 578K automated build entries from MSR challenge dataset [3] for our study. Since we are interested to investigate possible relationships between automated builds and peer code reviews, the collected entries should be associated with code reviews. In GitHub, pull requests are generally used for code reviews. Hence, we collect only such entries where each of the corresponding builds was triggered by a pull request (i.e., `gh_is_pr=true`) submitted by the developer. We also identify whether the commits in each

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.39

TABLE I
DATASET FOR EXPLORATORY STUDY

Build Status	Automated Build Only		Build + Code Review		Total	
	Entry	Project	Entry	Project	Entry	Project
Canceled	2,616	135	1,368	85	3,984	207
Errored	51,729	2,138	27,262	1,673	78,991	2,735
Failed	55,546	2,368	39,025	2,139	94,571	3,106
Passed	236,573	5,774	164,174	5,299	400,747	7,319

build underwent (i.e., `gh_num_pr_comments>0`) and did not undergo (i.e., `gh_num_pr_comments=0`) peer code reviews. We found 40% (i.e., 232K) of such commits reviewed by the developers. We extract automated build and test details (e.g., outcome, frequency) and code review activities (e.g., review comment statistics) from each of the collected entries for our comparative and inferential analysis. Table I shows the details of our collected dataset for the study.

III. ANSWERING RQ$_1$: AUTOMATED BUILD STATUS AND CODE REVIEW PARTICIPATION

To answer RQ$_1$, we divide automated build entries into two non-overlapping groups– builds with code reviews and builds without code reviews. The goal is to contrast between these two groups in terms of their build status and code review activities. An automated build can have one of these four statuses– *canceled*, *errored*, *failed* and *passed*. On the other hand, review comment counts from each entry could be considered as a proxy to code review participation. That is, if such a comment count is greater than zero, the commits associated with the build entry received code reviews and vice versa. Alternatively, one or more reviewers participated and the participation is denoted as "1" and vice versa. In GitHub, reviewers can submit two types of code review comments– *in-line comments* and *summary comments* – where they are also called as *pull request comments* and *issue comments* respectively. Thus, we consider one independent variable (i.e., automated build status) and two response variables (i.e., pull request comment count and issue comment count) for RQ$_1$, and perform statistical tests and further analyses to answer the research question in terms of a hypothesis as follows:

H1$_0$: Code review participation is not affected by the status of previous automated builds.
H1$_a$: Code review participation is significantly affected by the status of previous automated builds.

Test of Hypothesis: While 40% of the built commits from our dataset received one or more code reviews, developers did not participate in the code reviews of rest 60% commits from the same set of projects. Please note that 95%–100% of the projects from the MSR challenge dataset [3] adopt pull request based code reviews. Therefore, such lack of reviews might not be a mere coincidence and thus warrants an in-depth analysis. To ensure a fair comparative analysis, we pick up a random sample of 231,829 entries without code reviews (i.e., equal to the entries with reviews) from our dataset. Then we perform *Chi-squared tests* on the samples combining both groups, and investigate whether the code review participation is independent of corresponding build status or not. The test reported a *p-value* of 2.2e-16<0.05,

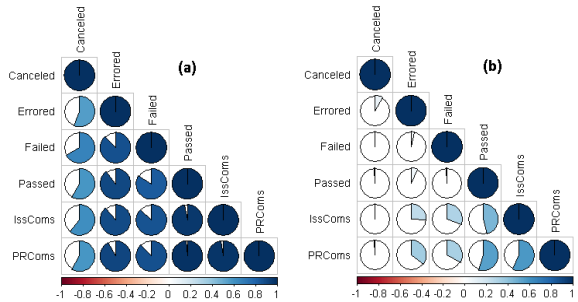

Fig. 2. Pearson correlation between build status and code review participation associated with (a) projects and (b) pull requests. IssComs=Issue comments, PRComs=PR comments

TABLE II
PULL REQUESTS WITH CODE REVIEW COMMENTS CHANGED

Previous Build Status	Issue Comments		PR Comments		All Review Comments		
	Add↑	Remove↓	Add↑	Remove↓	Add↑	Remove↓	Total↑↓
Canceled	15	21	9	7	20	24	65
Errored	448	198	232	122	**510**	**265**	**812**
Failed	1,379	676	610	299	**1,542**	**826**	**2,316**
Passed	3,711	1,388	2,048	791	**4,235**	**1,788**	**5,677**

↑ = One or more review comments added, ↓ = One or more comments removed from the reviews

which refutes the null hypothesis (H1$_0$). That is, alternative hypothesis (H1$_a$) is accepted, and code review participation is significantly affected by the status of automated builds.

Correlation Analysis: Although the alternative hypothesis (H1$_a$) is found true according to the above statistical test, we perform correlation analysis between independent and response variables for gaining further insights. In particular, we consider two entities with different abstraction levels– *project* and *pull request*, and calculate their build statistics and code review statistics. We calculate the number of *canceled*, *errored*, *failed* and *passed* automated builds for each entity while determining the frequency of their built commits that received code reviews later. The goal is to find out whether certain build status is correlated to code review participation or not. Fig. 2 shows the correlation plot between four build statuses and code review participation for (a) projects and (b) pull requests. We see that commits with *passed* builds received the maximum code reviews both in project level and in pull request level. That is, reviewers possibly get more confidence in reviewing such code that is syntactically correct and meets functional requirements, i.e., passes the automated builds and tests. However, as shown in Fig. 2, the *errored* and *failed* builds also triggered notable review participation.

Review Change Analysis: Review comment statistics for each entry in the challenge dataset are generally calculated for the time gap between the previous build and the current build on the same pull request [3]. That means, if such statistics are found changed during the current build submission, the previous build might have played a role given that corresponding logs reported the details of automated builds and tests performed. We thus analyze the automated build sequence (i.e., based on build starting time) of each pull request from our dataset, and determine their review comment changes. In particular, we identify the status of the immediate previous build and the change direction of review statistics in the current build for each of the pull requests. Table II shows the statistics of pull requests that underwent such review changes. We see that 28% (8,870) of 31,648 pull requests (i.e., associated with

TABLE III
CODE REVIEW COMMENTS OF TWO QUARTILES

Quartile	Issue Comments			PR Comments			All Review Comments		
	Mean	p-value	Δ	Mean	p-value	Δ	Mean	p-value	Δ
Q_1	0.60	<0.001*	0.35	0.24	<0.001*	**0.49**	0.84	<0.001*	**0.41**
Q_4	0.99			0.52			1.50		
Q'_1	0.62	<0.001*	**0.40**	0.32	<0.001*	**0.53**	0.94	<0.001*	**0.44**
Q'_4	0.97			0.54			1.51		

* = Statistically significant, Δ = Cliff's delta for effect size, Q'_i = Quartile for total build counts

code reviews) received further code reviews which might have triggered by the immediate previous builds. Passed builds are associated with most of these review changes (i.e., 18%) which confirms our findings from the correlation analysis. Besides, as shown in Table II, failed and errored builds also introduced a moderate amount (i.e., about 10%) of reviews. We also check three file change statistics for each of these pull requests from the dataset, and found that 99%–100% of them underwent file changes. That is, automated builds first trigger the code changes which in turn might warrant further code reviews.

Thus, to answer **RQ$_1$**, automated build statuses are very likely to affect code review participation. Passed builds encourage more code reviews than failed or errored builds. While a code review process is generally initiated by the patch submitter requesting peers for reviews, automated builds along with their outcomes might also trigger further code reviews.

IV. ANSWERING RQ$_2$: AUTOMATED BUILD FREQUENCY AND CODE REVIEW QUALITY

Our analyses in RQ$_1$ suggest that automated build status might affect code review participation while *passed* builds having the maximum influence. Our conjecture is that developers possibly felt comfortable in reviewing such code (i.e., investing their effort into) that has already gained a quality threshold (i.e., passed the automated builds and tests). However, too many and too frequent builds might introduce reviewing job overload [2] on the developers which is likely to hurt the review quality. We consider review comment counts as a *proxy* to code review quality [4, 6]. Thus, an investigation is warranted on how frequency of automated builds might affect the overall code review quality. We collect project level build statistics and review comment statistics, and contrast between the projects with high frequency builds and the projects with low frequency builds. We performed statistical tests and further analyses to answer the research question in terms of a hypothesis as follows:

H2$_0$: Quality of code reviews associated with highly frequent builds is similar to that of less frequent builds.
H2$_a$: Quality of code reviews associated with highly frequent builds is significantly higher than that of less frequent automated builds.

Test of Hypothesis: We calculate build frequency per month for each of the projects in our dataset. Then we divide them into four quartiles where first quartile (Q_1) contains the lowest 25% and fourth quartile (Q_4) contains the highest 25% of all the build frequencies. We collect the corresponding project entries from both quartiles, and compare their mean review comment counts– *issue comment counts/build* and *pull request comment counts/build*. We performed *Mann-Whitney Wilcoxon*

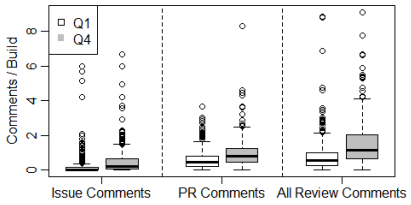

Fig. 3. Code review quality of projects from two different build frequency quartiles

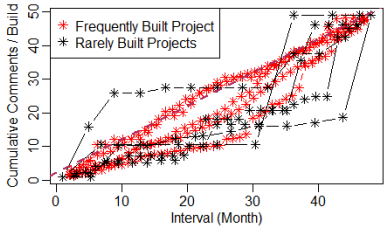

Fig. 4. Review comments of frequently built projects and rarely built projects

tests, and found that their review comment counts differ significantly, i.e., all *p-values* are less than 0.05. Table III reports the details of our statistical tests, and Fig. 3 shows the box plots of mean comment counts for the two quartiles. We see that code review quality (i.e., mean review comment count) is significantly higher for Q_4 (i.e., projects with frequent builds) than Q_1 (i.e., projects with less frequent builds) in terms of all measures– *mean*, *p-value* and *Cliff's delta (effect size)*– which refutes the null hypothesis (H2$_0$). We also repeat the same experiments by considering total build counts rather than frequency per month for each of the projects, and reached at the same conclusion. Thus, build frequency has a significant impact on the code review quality of the projects. Since automated tests are performed simultaneously with automated builds and thus have the same frequencies, the above findings regarding code review quality also equally apply to them.

Interval-Aware Comparative Analysis: While the above statistical analysis shows that the alternative hypothesis (H2$_a$) is true, we performed further analysis to gain more insights. In particular, we examine and compare the code review statistics of frequently built projects (i.e., fourth quartile, Q_4) and rarely built projects (first quartile, Q_1) over specific time interval such as *month*. We select Top-5 projects from Q_4 and Top-5 projects from Q_1 where each of the projects is 3–4 years old. Then, we calculate review comments/build for each of the 48 months for the projects, and plot the cumulative comments/build for each project. From Fig. 4, we see that the cumulative curves for the frequently built projects are close to linear with an upward slope. This suggests that these projects maintained steady review activities over the time period in question. On the other hand, the curves for the less frequently built projects are zigzag and do not show a regular structure. That is, they failed to maintain the regular code review activities, which is also manifested by their declining review comment statistics.

Thus, to answer **RQ$_2$**, automated build frequency is very likely to have an impact on the quality of code reviews in the open source projects. While frequent builds help maintain an acceptable code quality standard for the projects, they also help trigger more code reviews than the infrequent builds.

TABLE IV
PERFORMANCE OF PREDICTION MODELS

Algorithm	Metrics	Overall Accuracy	New Review Triggered	
			Precision	Recall
Naive Bayes	{all metrics}	58.03%	68.70%	29.50%
	{build status, code review}	56.50%	78.60%	17.80%
Logistic Regression	{all metrics}	60.56%	64.50%	47.00%
	{build status, code review}	60.18%	64.30%	45.60%
J48	{all metrics}	**64.04%**	**69.50%**	**50.10%**
	{build status, code review}	62.64%	73.80%	39.20%

V. ANSWERING RQ_3: AUTOMATIC PREDICTION OF NEW CODE REVIEWS USING A MODEL

Automated builds and tests often introduce source code changes which in turn might or might not trigger further code reviews. According to our analysis, about 100% of the builds from the dataset are associated with code level changes. However, code reviews were triggered for only 40% of the pull requests (details in Section II). Automated identification of such pull requests and their builds beforehand could help the project developers or stake holders with important decision making on code reviews. Thus, a prediction model is warranted which can automatically predict whether an automated build is likely to trigger new code reviews or not.

We develop a prediction model where the model is trained on build log data using three machine learning algorithms. In particular, we use automated build status– `tr_status`, three code change statistics–`gh_diff_files_added`, `gh_diff_files_deleted` and `gh_diff_files_modified`, two test change statistics– `gh_diff_tests_added` and `gh_diff_tests_deleted` and two review comment statistics for the pull request–`gh_num_issue_comments` and `gh_num_pr_comments` – as the predictor variables. The response variable of each current build is determined based on whether the next build entry (i.e., based on building date and time) for the same pull request has a *changed review comment statistic* or not. Thus, the response variable for each of our build entries takes one of these two values–*"new review"* or *"unchanged."* We used a randomly sampled set of 26,516 build entries from our dataset for the experiments where equal number of instances from both classes are ensured. We use *Naive Bayes (NB)*, *Logistic Regression (LR)* and *J48* from WEKA [1] workbench for the training, and apply 10-fold cross validation for the testing of our prediction models. Table IV summarizes our findings.

From Table IV, we see that J48-based model performed the best in separating review triggering build entries from the rest. Our model classifies the build entries with 64% overall accuracy which is promising as a proof of concept. Besides, the model can identify the true-positives with about 70% precision and 50% recall which are also promising. Such findings clearly answer our third research question regarding automatic prediction on code review triggering–$\mathbf{RQ}_3$.

VI. DISCUSSION & CONCLUSION

In this paper, we report an exploratory study using 578K automated build entries from MSR challenge dataset, where we investigate the impact of continuous integration on code reviews. We explore two different aspects of continuous integration–*automated build status* and *automated build frequency* and two aspects of code review–*review participation* and *review quality*, and investigate how the former aspects might affect the later. We apply several statistical tests, and perform correlation and comparative analysis to answer our three research questions. Our findings both confirm intuitive beliefs and reveal new meaningful information as follows:

Automated build status has a notable impact on code review participation where passed builds having the maximum influence. Our analyses show that automated builds triggered new code reviews for 28% of the 31,648 pull requests from our dataset. While *passed* builds played the major role, *errored* and *failed* builds also brought about new code changes and thus, triggered further code reviews for 10% of the pull requests.

Build frequency has a significant impact on the quality of code reviews in the open source projects. Our analyses suggest that code review comments are significantly higher for frequently built projects than that of rarely built projects. Frequently built projects often maintain a steady level of code review activities over the years, which is most probably missing from the rarely built projects according to our findings.

Experiments with the three prediction models show that most of our identified metrics– build status, code change statistics, test change statistics and review comment statistics– are effective in predicting whether an automated build might trigger new code reviews or not. Since the dataset was skewed, we conduct experiments with a randomly sampled subset that contains equal number of instances from both classes. Our model provides a reasonable accuracy of 64% with up to 70% precision and 50% recall. Although peer code reviews are reported to be effective for quality improvement of software systems, they are generally done manually and are often time-consuming. Our model can offer automatic supports in the code reviews and software quality management by identifying the appropriate pull requests for code reviews using build logs.

Acknowledgement: This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] WEKA. URL http://www.cs.waikato.ac.nz/ml/weka/.
[2] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. The influence of non-technical factors on code review. In *Proc. WCRE*, pages 122–131, 2013.
[3] M. Beller, G. Gousios, and A. Zaidman. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *Proc. MSR*, 2017.
[4] A. Bosu, M. Greiler, and C. Bird. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *Proc. MSR*, pages 146–156, 2015.
[5] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey. Investigating code review quality: Do people and participation matter? In *Proc. ICSME*, pages 111–120, 2015.
[6] O. Kononenko, O. Baysal, and M. W. Godfrey. Code Review Quality: How Developers See It. In *Proc. ICSE*, pages 1028–1038, 2016.
[7] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proc. MSR*, pages 192–201, 2014.
[8] R. Morales, S. McIntosh, and F. Khomh. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. In *Proc. SANER*, pages 171–180, 2015.
[9] S. Panichella, V. Arnaoudova, M. D. Penta, and G. Antoniol. Would Static Analysis Tools Help Developers with Code Reviews? In *Proc. SANER*, pages 161–170, 2015.
[10] P.C. Rigby, B. Cleary, F. Painchaud, M. Storey, and D.M. German. Contemporary Peer Review in Action: Lessons from Open Source Development. *TSE*, 29(6): 56–61, 2012.