A Study on the Interplay between Pull Request Review and Continuous Integration Builds

Fiorella Zampetti
University of Sannio
Benevento, Italy

Gabriele Bavota
Università della Svizzera Italiana
Lugano, Switzerland

Gerardo Canfora
University of Sannio
Benevento, Italy

Massimiliano Di Penta
University of Sannio
Benevento, Italy

Abstract—**Modern code review (MCR) is nowadays well-adopted in industrial and open source projects. Recent studies have investigated how developers perceive its ability to foster code quality, developers code ownership, and team building. MCR is often being used with automated quality checks through static analysis tools, testing or, ultimately, through automated builds on a Continuous Integration (CI) infrastructure. With the aim of understanding how developers use the outcome of CI builds during code review and, more specifically, during the discussion of pull requests, this paper empirically investigates the interplay between pull request discussion and the use of CI by means of 64,865 pull request discussions belonging to 69 open source projects. After having analyzed to what extent a build outcome in uences the pull request merger, we qualitatively analyze the content of 857 pull request discussions. Also, we complement such an analysis with a survey involving 13 developers. While pull requests with passed build have a higher chance of being merged than failed ones, and while survey participants confirmed this quantitative finding, other process-related factors play a more important role in the pull request merge decision. Also, the survey participants point out cases where a pull request can be merged in presence of a CI failure, *e.g.,* when a new pull request is opened to cope with the failure, when the failure is due to minor static analysis warnings. The study also indicates that CI introduces extra complexity, as in many pull requests developers have to solve non-trivial CI configuration issues.**

Index Terms—**Continuous Integration; Pull Requests; Modern Code Reviews**

I. Introduction

Modern Code Review (MCR) is nowadays a widely adopted practice in industrial [1], [7] and open-source [4], [14], [20], [30], [31] projects. MCR originates from the more formal and rigorously defined idea of code inspection [11], in which developers physically meet and perform a checklist-based code reading and discussion. MCR makes this process less formal and introduces a tool support for distributed code reviews.

Often, the MCR process is accompanied by automated quality assessment tasks, such as the execution of static analysis tools [27], or a verification activity performed executing test cases accompanying the patch to be reviewed [36]. This activity is put to its extreme when a Continuous Integration (CI) process is adopted. CI features the automated execution of builds on a dedicated infrastructure every time a change is being pushed.

Some hosting infrastructures integrate CI with code review, so that developers can exploit the output of the CI builds to drive (part of) their reviews. In particular, GitHub allows the integration of hosted or internal CI solutions such as Travis-CI, Circle-CI or Jenkins within the PR review process.

Previous work has shown how CI improves the overall quality of a project [36], [41]. However, the actual interplay between executing automated builds on a CI server and reviewing a patch (or a PR) remains unexplored. Vasilescu *et al.* [36] investigate how CI is being used to test a PR on a separate branch, but nowadays, software projects have put in place a tighter interplay between CI, executed since a PR is opened and during the PR discussion, and code review.

The goal of this paper is to empirically investigate how developers leverage CI outcome (and build logs) during a code review process. More specifically, we analyze the process of PR reviewing in 69 open source projects hosted on GitHub and using Travis-CI as CI infrastructure.

As a first step, we looked at the build status when a PR was opened, and when it was closed, to determine whether such a status correlates with the chance of merging or rejecting the PR. We also looked at the number and status of builds executed during the PR review. Finally, we determine the extent to which CI build-related factors contribute to existing PR acceptance prediction models, in particular, the model by van der Veen *et al.* [34], [35]. After that, we qualitatively analyzed a statistically significant sample of 857 PRs that broke the build when submitted, to investigate the extent to which different kinds of build failures generate a PR discussion, and how this discussion contributes to the PR acceptance. As an output of this analysis, we produce a detailed taxonomy of build failures being discussed in PRs (see Fig. 2), and of the nature of such discussions. Finally, we complemented this qualitative analysis with a survey, to which 13 developers participated.

The study results highlighted that while PRs with passed builds have slightly more chances of being merged than when builds are broken, other process-related factors have a stronger correlation with such a merger. Nevertheless, 11 out of 13 respondents highlighted that the build outcome contribute to the PR merger and the majority of them do not accept a PR if the build is failing. However, they also pointed out some exceptions to the latter rule that are reflected also in our quantitative results (*e.g.,* cases in which the build status is ignored since it is related to minor warnings raised by static analysis tools).

Replication Package. The data used in our study is available for replication purposes [39].

II. Study Design

The *goal* of this study is to investigate how information from CI builds (build status and content of build logs) is taken into account during PR discussion, in order to decide whether the PR should ultimately be merged or closed. The *perspective* is of researchers interested to understand the adoption practices of CI, also in the context of code reviews. The study *context* consists of 69 open source projects hosted on GitHub and using the Travis-CI infrastructure, and of 13 industrial developers participating in a survey.

The first piece of information a developer receives from CI is the build status. When a PR is opened, CI is triggered and quality assessment tasks — including testing and static analysis — are performed. A successful (*i.e.,* green) build could constitute already a positive input for the reviewers. If not, the code could be modified during the PR discussion, in order to make the build pass. We investigate how this happens by answering our first research question:

> **RQ**$_1$ *How does the CI build status relate to PR merging?*

If the build is not passing, the build failure may or may not be discussed as part of the PR discussion, and reviewers may bring up different pieces of information. First, they can refer to the observed behavior, *i.e.,* what is visible from the build log. After that, they can start discussing the possible root cause(s) of the problem. Finally, they can propose one or more solutions, recommending changes that generate new builds, which may or may not pass. We study reviewers behavior in discussing PRs by answering our second research question:

> **RQ**$_2$ *When a PR generates a build failure, what problems do developers discuss?*

A. Context Selection and Data Extraction

The *context* consists of 69 open source projects hosted on GitHub that use Travis-CI. Moreover, in order to assess the synergies between CI and pull-based development, we have surveyed 13 CI practitioners, selected based on personal contacts of the authors.

The 69 projects we studied have been selected as follows. We used the GitHub API in order to sort the whole set of GitHub projects in terms of popularity computed considering their number of stars. After that, we analyzed the projects from the most to the least popular, filtering out those that did not use Travis-CI. More specifically, we cloned the project repository locally and searched for the presence of the *.travis.yml* file. Then, using the Travis-CI API, we filtered out projects that did not have any build executed as a consequence of a PR event. Note that we only considered PRs having at least one comment that has not been posted by a bot. This was done using the GitHub API and matching the pattern `[bot]` in the field *Type*, or, in other cases, by matching the same pattern to the *Login* field. The search ended once we reached a list containing 100 popular projects using Travis-CI to evaluate the code contribution submitted as a PR to the project. Finally, we have filtered out projects that had less than 100 build failures related to PRs submission, leading to the 69 studied projects.

To extract the data needed for our analysis, we first downloaded and extracted information about each build using the Travis-CI API. More specifically, we looked at the build status (passed, failed, errored or canceled), the type of event that has generated the build (*e.g.,* push, pull request), the commit ID related to the event along with its message, the number of build jobs with their identifiers and, finally, the link to the change in the GitHub repository. The latter is needed in order to retrieve the PR number related to the build under analysis.

After that, we downloaded the log produced by Travis-CI during the build process and, by means of regular expressions, we checked the ending status of each job. Since Gallaba *et al.* [13] pointed out that sometimes build passes because of an abuse of the *allow_failure* (allow ignoring a failed job), we considered a build failed even if a job fails (when the *allow_failure* is set) while the build passes. Finally, we discarded canceled builds, while we treated failed and errored builds similarly (both considered as not passing).

Based on the collected data, we were able to reconstruct the history in terms of builds status for each PR in our dataset. However, this is not enough to answer **RQ**$_1$. We also needed to extract the factors impacting the PR merger as reported by van der Veen *et al.* [34], [35]. Starting from the PRs number, we used the GitHub API to download the detailed information belonging to each of them, *e.g.,* the number of changed lines, the number of files involved, the number of comments.

As a result, we obtained a PR dataset made up of 189,700 pull requests, out of which 56,881 were closed without being merged, and 132,819 ended with a merge operation. Finally, in the overall dataset, we had a total of 64,865 PRs that broke the CI process during their initial submission (*i.e.,* creation).

B. Analysis Methodology

To address **RQ**$_1$, we first check a quite "expected" conjecture, *i.e.,* whether PRs are more likely to be merged when the build passes. Although this seems to be obvious, there may be still many cases in which the build status is ignored, *e.g.,* it is related to a static analysis false positive, to a flaky/unaligned test, to a change belonging to a different branch that has to be merged, etc. We use the Fisher s exact test and the Odds Ratio (OR) [12] to compare the odds passed builds have to be merged against broken builds. We perform this analysis considering the build status (i) when the PR was opened, and (ii) upon its merger/closure.

Since the reasons behind a PR merger can clearly go beyond the build status, we analyze how a number of factors — related to the CI process and to the overall development process — correlate with the merger. As build-related factors, we consider: (i) the number of builds over the PR discussion; (ii) the percentage of failed builds during the PR discussion over the total number of builds; (iii) the status of the first build, *i.e.,* when the PR is submitted; and (iv) the status of the last build, *i.e.,* when the PR is merged or closed. We combine these factors with process-related factors correlated by van der Veen

TABLE I
FACTORS WE CORRELATE WITH PR MERGER: BUILD-RELATED FACTORS AND PROCESS FACTORS DEFINED BY VAN DER VEEN *et al.*.

PROCESS FACTORS [34], [35]	
Factor	**Description**
Age	PR duration from its creation
Contribution Rate	% of authors commits before the PR
Accept Rate	% of previously merged PRs from the same author
Additions	# of lines added in the PR commits
Deletions	# of lines deleted in the PR commits
Commits	# of commits included in the PR
Changed Files	# of files changed upon opening the PR
Comments	# of discussion comments
Review Comments	# of comments in the PR attached to code
Core Member	whether the PR author is a project member
Intra-Branch	whether source and target PR repos match
Contains Fix	whether the PR aims at fixing an issue
Last Comm. Mention	whether the last commit mentions a user
Has Test Code	whether test cases are included in the PR
BUILD-RELATED FACTORS	
Factor	**Description**
PR Builds	# of builds over the PR discussion
Failed Builds	% failed builds during the PR discussion
First Build Status	build status when submitting the PR
Last Build Status	build status upon merging/closing the PR

et al. [34], [35] with PR mergers and used to prioritize PRs. The complete list of factors is reported in Table I.

To analyze how such factors (*i.e.,* process-related factors and build-related factors) correlate with a PR merger, we build machine learning predictors on such factors considering the merger as dependent variable. To understand the role played by build-related factors we (i) build models with and without considering these factors, and (ii) report indicators of the importance of such factors in the machine learning models.

Before applying the machine learners, we identify groups of factors that correlate and, for each group, we only keep one of them. To this aim, we use the *R varclus* function of the *Hmisc* package [18], producing a hierarchical clustering of features based on their correlation, in turn computed with a specified correlation measure (we use the Spearman s ρ rank correlation). Then, we identify clusters by cutting the tree at a given level of ρ^2 that we set at $\rho^2 = 0.64$, which corresponds to a strong correlation (*i.e.,* $\rho = 0.8$) [9]. After that, we remove features that in our dataset do not vary or vary too much, because they would not be useful to build a predictor. This is performed using the *RemoveUseless* filter implemented in Weka [17], which removes from a dataset, features that never vary as well as features with a percentage of variance above a threshold (we set to 99%). Finally, to deal with unbalanced data (in terms of merged and not merged PRs), we try to apply under-sampling on the majority class by means of *SpreadSubSample* filter implemented in Weka [17]. However, as explained in Section III-A, we report results without re-balancing as it does not improve the results.

Then, we build the machine learning models on the training set and use them to perform predictions on the test set. We experiment with five different machine learners implemented in Weka [17]: Decision Trees (J48), Bayesian classifiers, Random Forests, Random Trees, and Bagging with Decision Trees. We used such machine learners with their default configuration.

We run the models using a 10-fold cross validation over the PR dataset and we report the Precision, Recall, Area Under the Receiver Operating Characteristics curve (AUC), and Matthews Correlation Coefficient (MCC) [23] for both prediction categories (*i.e.,* merged and not merged). We use MCC since it is a measure used in machine learning assessing the quality of a two-class classifier especially when the classes are unbalanced. It ranges between -1 and 1 (0 means that the approach performs like a random classifier) and it is defined as:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(FN+TN)(FP+TN)(TP+FN)}}$$

As for other correlation measures, $MCC < 0.2$ indicates a low correlation, $0.2 \leq MCC < 0.4$ a fair, $0.4 \leq MCC < 0.6$ a moderate, $0.6 \leq MCC < 0.8$ a strong, and MCC ≥ 0.8 a very strong correlation [9]. More importantly, we report information about the importance of the considered factors using the Mean Decrease Impurity (MDI) [22] (averaged over the cross-validation folds), which measures the importance of variables on an ensemble of randomized trees.

To address **RQ**$_2$, we randomly selected from the set of 64,865 build failures in our dataset a statistically significant sample ensuring a 99% confidence level $\pm$ 5%. The build failures in our sample have been chosen proportionally to the number of PRs belonging to each project. When for a project the overall number of PRs to include in the sample was lower than 10, we oversampled and took 10 PRs from the project. This resulted in the selection of 857 discussions that we manually analyzed to characterize the problems discussed by the developers when a PR generates a build failure.

The manual analysis was conducted to classify the PR discussions as false positives (*i.e.,* the discussion is unrelated to the build failure) or assigning a set of "labels" describing (i) the issue: the build failure issue subject of the discussion (*e.g.,* a test case failed), (ii) the root cause triggering the issue identified in the discussion (*e.g.,* a bug in the application logic), and (iii) the solution discussed/adopted (*e.g.,* the bug has been fixed). Each of these labels was optional. For instance, it is possible that in the PR comments only the build failure issue is mentioned, without discussing possible solutions.

For each type of label (*i.e.,* issue, root cause, solution), we started from a predefined list of categories that we defined running a pilot study in which the three authors manually analyzed a random set of 30 PRs triggering build failures. These 30 PRs are not part of the set of 857 PRs we use to answer RQ$_2$. Each of the 30 PRs was analyzed by each of the authors and, in the end, an open discussion was performed to agree on the procedure and on the categories to adopt.

The labeling procedure was supported by a shared Google sheet and has been conducted as defined in the following. The initial sample of 857 PRs has been divided into two different sets (400+457). The first author labeled all the 400 discussions belonging to the first group, while the second and third author acted as second evaluator for half of the discussions each (200 each). Thus, every discussion belonging to the first set was labeled by two authors. Each author independently analyzed the discussions assigned to her, selecting the elements

TABLE II
SURVEY DESIGN (EXCLUDING DEMOGRAPHICS).

GENERAL QUESTIONS
Does the status of the build influence the merger of a PR?
If the last build during the PR discussion fails, do you still merge the PR?
What are the reasons for accepting the PR even in presence of a build failure?
QUESTIONS ASKED FOR EACH BUILD FAILURE CATEGORY AT THE FIRST-LEVEL OF FIG. 2.
How often do you discuss in the code review process the problems described above?
Please rate the usefulness of discussing different kinds of build failures in the context of PR review.
What are the actions performed during PR review in order to make the build successful or to isolate/locate the problem?

being discussed and categorizing them using the predefined categories or defining a new one when needed. To assign the labels and categories to a discussion the author inspected the whole PR discussion as well as the build failure log, in order to obtain contextual information needed to better understand the discussion. Every time an author had to label a discussion, the Google sheet also showed the list of categories created so far, allowing her to select one of the already defined categories. This means that a new category defined by author a_1 is visible to author a_2 that, however, does not know where this label was used. In a context like the one in this work, where the number of possible categories (*e.g.,* types of build failure issues) is extremely high, such a choice helps using consistent naming while not introducing a substantial bias.

At the end of the first round of the labeling process, an open discussion was performed among the authors in order to solve conflicts. A conflict in this scenario can happen for three reasons. First, only one of the two authors could label the discussion as a "false positive". Second, the two authors could have assigned different labels (*i.e.,* elements being discussed) to the same discussion (*e.g.,* the first author labeled the issue discussed while the second author labeled the root cause). Third, assuming that the same labels are selected (*e.g.,* both labeled the type of issue discussed), the category value could be different (*i.e.,* two different issues are discussed accordingly to the two authors). For these reasons, we had a quite high number of conflicted artifacts (144 out of 400 — 36%). Once the agreement on the procedure has been reached, the first author worked in isolation to run the second round of the labeling process, accounting for the remaining 457 PRs discussions.

The qualitative analysis of a statistically significant sample of PR discussions could be insufficient to unveil the CI build failures developers scrutinize when assessing a PR and, also, how CI is used during a PR review process, *e.g.,* whether developers debug a PR by performing subsequent CI builds or whether this is done on their local machines. To collect additional data, we conducted a survey in which we asked developers how they use CI when reviewing PRs. The survey questionnaire was made up of the following sections: (i) Demographics questions which asked for the highest degree, development experience, current position, domain in which the developer works, and CI technology being used. (ii) More specific expertise questions investigating the knowledge about CI pipeline usage and management, and code reviewing. (iii) Questions about the relationship between CI builds and merge of a PR (second part of Table II). (iv) Questions about the extent to which different kinds of build failures are discussed, whether the discussion has the purpose of determining the PR merger, and what the actions performed to aid the PR resolution are (example of available answers are: nothing, local build, re-execute the build on the CI server). Respondents had to provide an answer to these question for each category of build failure being discussed that emerged from the first part of the study (qualitative analysis of PRs).

The survey questionnaire has been made available through Google Forms, and shared through personal contacts of the authors. We received 13 responses from professional developers (one of which product owner and another pipeline configurator) working in different domains, including financial, system integration, consultancy, travel, and digital identity management. Their expertise in configuring a CI pipeline was low in 2 cases, moderate in 5 cases, high in 6 cases. Instead, the expertise in pull-based development and code review was low in 2 cases, moderate in 4 cases, and high in 7 cases. Finally, respondents reported the use of different CI environments, in some cases multiple ones: Jenkins (7), GitLab (4), Bamboo (3), Concourse, TeamCity and Azure DevOps (one each).

III. STUDY RESULTS

This section presents and discusses results aimed at addressing the research questions formulated in Section II.

A. *How does the CI build status relate to PR merging?*

Out of the 189,700 studied PRs, 109,889 (57.9%) were only built once, *i.e.,* when the PR was created. Further 32,918 (17.4%) were built twice, and only the remaining 24.7% underwent multiple builds, with a long tail of outliers: the minimum, first and second quartile is 1 build, the third quartile is 2 builds, and the maximum is 256.

A deeper analysis of the cases having a high number of build failures, highlighted two scenarios. There are PRs aimed to release a new version of the software (PR #7033 in *meteor/meteor*[1]) in which the description states: *"This pull request is an easy way to collect changes that will be going into Meteor 1.3.3 and run tests against them in an isolated, consistent environment (CircleCI and TravisCI)"*. In these cases, each commit part of the release generates a new build on the CI server, and only when all the commits successfully build the PR is merged and the new version is released. Moreover, there are cases in which, during the PR discussion, the integrator

[1]To access the PR body use https://github.com/$owner/$repo/pull/$PR as https://github.com/meteor/meteor/pull/7033

TABLE III

RELATIONSHIP BETWEEN PR OUTCOME AND BUILD STATUS (I) WHEN OPENING THE PR (II) WHEN CLOSING/MERGING IT.

Status when opening the PR		
	Merged	Not merged
Passed	101,513	38,654
Failed	31,305	18,228
Status when closing/merging the PR		
	Merged	Not merged
Passed	113,218	43,775
Failed	19,600	13,107

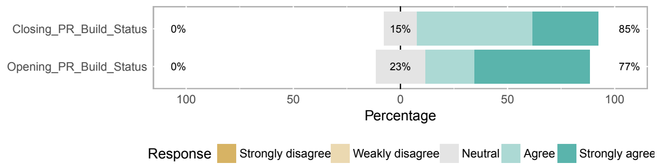

Fig. 1. Survey: Impact of build status on the PR merging decision.

points out problems that must be addressed for having a chance of a PR merger (for instance PR #503 in *moment/moment*). In the latter case, the contributor starts to modify her change following the comments pointed out by the integrator.

The upper part of Table III reports the confusion matrix of the number of PRs that caused a build pass and failure upon opening them and, in each case, they were either merged or closed. Fisher s exact test indicates that passed PRs have significantly more chances of being merged than those that failed (p-value$<$ 0.001), with an OR=1.5. Similarly, if we analyze the build status when the build was closed or merged (Table III bottom part), again we obtain a statistically significant difference (p-value$<$ 0.001), with an OR raising to 1.72. In both cases, although it is clear also from the test results that PR with passed builds have significantly more chances of being merged, we can still notice a conspicuous percentage of "outlier" PRs. Above all, both tables show a large number of PRs being closed although the build passes (both at the beginning and at the end of the discussion), indicating that, as expected, there are other reasons for PR outcome beyond the build status itself. At the same time, Table III also shows how 31,305 successfully merged PRs are opened with a failed build. Nevertheless, only 19,600 of them remain in the failed status at the end of the discussion. The remaining 11,705 have their build status fixed during the PR discussion.

As explained in Section II-B, we complemented the analysis of PRs with a survey. Fig. 1 reports responses to the general questions *i.e.,* whether developers agree that (i) the build status when opening the PR and (ii) the last build status, would influence the decision upon merging the PR. Respondents also indicated the reasons why a PR could still be merged even if the build fails: a new PR is opened in the meantime to address the failure (5 responses), the failure (*e.g.,* static analysis warning) is considered irrelevant (5 responses), the failure is not related to the PR change (5 responses), or the failure is transient (1 response).

TABLE IV

RESULTS OF THE RANDOM FOREST PREDICTORS ON PR MERGER.

Build-Related Feat.	TP Rate	FP Rate	Precision	Recall	MCC	AUC
PREDICTION OF NOT-MERGED PRs						
Yes	0.53	0.08	0.74	0.53	0.50	0.82
No	0.51	0.07	0.75	0.51	0.50	0.82
PREDICTION OF MERGED PRs						
Build-Related Feat.	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**MCC**	**AUC**
Yes	0.92	0.47	0.82	0.92	0.50	0.82
No	0.93	0.49	0.82	0.93	0.50	0.83

TABLE V

MEAN DECREASE IMPURITY (MDI) OF FACTORS USED TO PREDICT PR MERGER (BUILD-RELATED FACTORS ARE SHOWN IN ITALIC).

Factor	MDI
Core Member	0.394
Contains Fix	0.349
Age	0.229
Last Build Status (ends failing)	0.101
First Build Status (starts failing)	0.088
Failed Builds	0.046
Changed Files	0.027
Commits	0.026
Review Comments	0.024
Comments	0.024
PR Builds	0.017
Has Test Code	0.014
Deletions	0.012
Last Comm. Mention	0.005
Contribution Rate	0.002
Accept Rate	0.002

PR opened with passed CI builds have 1.5 more chances of being merged than those failing. Such chances increase up to 1.72 if the build passes at the end of the PR discussion. The survey respondents indicate that build should pass to merge a PR, but also point out cases in which this is not needed.

After having obtained an overview of the extent to which the build status correlates with the chance of merging a PR, we combined PR-related factors with the process factors considered by van der Veen *et al.* [34], [35]. Note that two process-related factors were filtered out during the preprocessing, *i.e., additions* and *intra_branch*. The former because highly correlates with the number of changed files in the PR, while the latter never changes in the whole PR dataset.

Results of merger prediction using Random Forests are reported in Table IV with and without build-related factors. Since Random Forests outperform other predictors, the latter are omitted. Also, we report results without re-balancing (undersampling) as it worsens the models performance in both cases. As the table shows, (confirming what done by van der Veen *et al.* [34], [35]) it is possible to use process-related factors to predict a PR merger. Overall, the model achieves a good precision especially when predicting merged PRs, and the recall is particularly high when predicting merger. Also the AUC (0.82) is satisfactory.

At the same time, we can notice how adding/removing build-related factors, in essence, does not change the model s prediction performance. This seems to indicate that, while having a passed build matters when deciding whether merging a PR or not, there might be more important factors influencing such a decision.

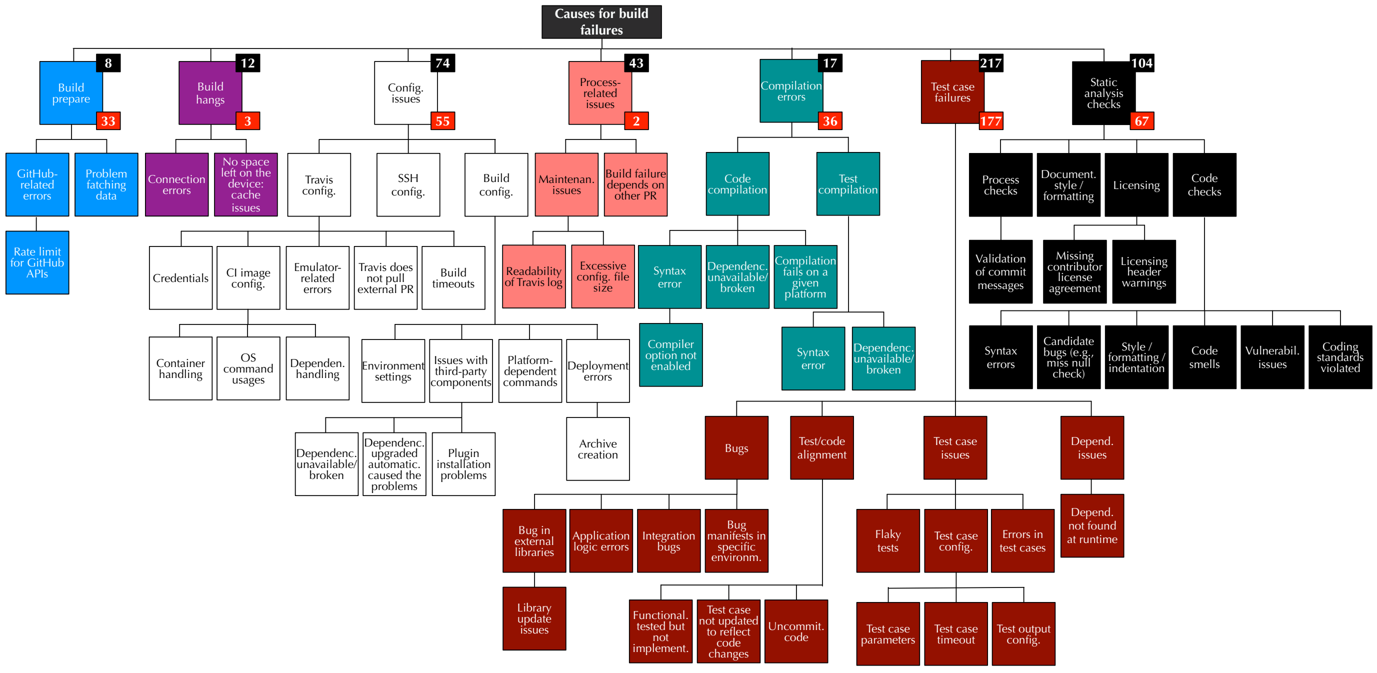
Causes for build failures
8
Build prepare
33
12
Build hangs
3
74
Config. issues
55
43
Process-related issues
2
17
Compilation errors
36
217
Test case failures
177
104
Static analysis checks
67
GitHub-related errors
Problem fatching data
Rate limit for GitHub APIs
Connection errors
No space left on the device: cache issues
Travis config.
SSH config.
Build config.
Credentials
CI image config.
Emulator-related errors
Travis does not pull external PR
Build timeouts
Container handling
OS command usages
Dependen. handling
Environment settings
Issues with third-party components
Platform-dependent commands
Deployment errors
Dependenc. unavailable/ broken
Dependenc. upgraded automatic. caused the problems
Plugin installation problems
Archive creation
Maintenan. issues
Build failure depends on other PR
Readability of Travis log
Excessive config. file size
Code compilation
Test compilation
Syntax error
Dependenc. unavailable/ broken
Compilation fails on a given platform
Compiler option not enabled
Syntax error
Dependenc. unavailable/ broken
Bugs
Test/code alignment
Test case issues
Depend. issues
Bug in external libraries
Application logic errors
Integration bugs
Bug manifests in specific environm.
Library update issues
Functional. tested but not implement.
Test case not updated to reflect code changes
Uncommit. code
Flaky tests
Test case config.
Errors in test cases
Test case parameters
Test case timeout
Test output config.
Depend. not found at runtime
Process checks
Document. style / formatting
Licensing
Code checks
Validation of commit messages
Missing contributor license agreement
Licensing header warnings
Syntax errors
Candidate bugs (e.g., miss null check)
Style / formatting / indentation
Code smells
Vulnerabil. issues
Coding standards violated

Fig. 2. Taxonomy of PR discussions.

Table V reports the MDI for the considered factors, ranked in order of importance. As the table shows, the first three factors (not related to the build status) exhibit an MDI much larger than the remaining ones. Clearly, PRs belonging to a core member have a higher likelihood of being merged, and so PRs related to bug fixes. As already pointed out by van der Veen *et al.* [34], [35], the PR age matters, in the sense that recent PRs have more chances of being merged than older ones (as time passes, the merging likelihood decreases). We can also notice that, although some build-related factors (and in particular those related to the build status at the end and at the beginning of the PR discussion) are ranked high, they have (in terms of MDI) less influence than the three process-related factors previously mentioned.

While build-related factors — especially those related to the build status when the PR is opened and when it is closed/merged — are ranked high by PR merger predictors, their role/effect on the predictor performance is negligible with respect to process-related factors previously considered by van der Veen *et al.* [34], [35]. Instead, the number of builds positively correlates with the merger.

B. When a PR generates a build failure, what problems do developers discuss?

Fig. 2 shows the taxonomy of build failures being discussed in PRs. We defined seven high-level categories, in turn, detailed into sub-categories. The number reported in the up-corner of each category refers to the number of PRs discussed, while the one at the bottom refers to the not discussed ones. While (not surprisingly) most of the build failures are due to testing or static analysis, we can also notice how process-related issues or cases of build hangs are frequently discussed when they occur. This is less likely for build prepare or compilation failures.

TABLE VI
TYPE OF INFORMATION DISCUSSED BY CATEGORY.

	Description	Root Cause	Solution	Patch
Build hangs	12	6	2	0
Build prepare	8	3	1	0
Compilation errors	17	9	7	4
Configuration issues	74	58	29	9
Process-related issues	43	23	14	6
Static Analysis checks	101	54	71	57
Test case failures	212	154	101	68
Overall	467	307	225	144

Table VI reports the typical content of a PR discussion for different categories. In most cases, at the minimum, the discussion provides a short description of the problem and, sometimes (more often for testing and configuration problems) the problem root cause. In some cases, a solution (and even a patch) is provided, and this happens more often for static analysis problems, but also for about half of the testing, configuration, and compiling errors.

In the following, we qualitatively explain each category of build failure we found discussed in PRs, by also providing examples and by reporting specific feedback and insights provided by our survey participants.

Test case failures may be due to different kinds of problems, including (i) the presence of bugs in the production code and/or in third-party libraries, (ii) the way in which the developers have configured/executed the test suite, *e.g.,* flaky tests or wrong configuration of the test suite to be executed, (iii) a misalignment between production and test code, and (iv) missing/wrong dependencies only identified at runtime (*i.e.,* not discovered by the compiler, if any).

Those kind of problems are the ones that are highly discussed by contributors once having submitted a PR that ends with a build failure. This result is consistent with the respondents to our survey. Indeed, only 2 out of 13 respondents stated that test case failures in CI are only occasionally discussed, while 8 of them report that the discussion is useful for the PR merger. Moreover, when trying to isolate and address those kinds of failures, 7 respondents use the private builds, while other 4 prefer to re-execute the build on the CI server.

Going more in-depth on the type of highly discussed failures, we start from those cases due to bugs related to third-party libraries, often less obvious to discover than production code bugs. A more interesting and discussed problem is the one in which the bug that results in a test case failure belongs to an external library that has been used in order to add/optimize a functionality. Consider as an example the PR #1409 from the *scrapy/scrapy* project in which the contributors argue about an undesired behavior due to a new release of a library. Specifically, one comment states that *"it is probably a bug introduced by some dependency upgrade [...] Twisted 15.3.0 was released yesterday for example"*, while another one highlights the possible root cause: *"[...] it can be related to twisted/twisted@a8d8a0c - see this line."*.

It may also happen that a test suite fails only in a specific environment. Consider the PR #4877 belonging to *serverless/serverless* in which the test failures occur on a specific version of "Node.js". One comment mentions: *"It actually crashes [...] due to usage of [].includes which Node v4 doesn t support."* and also refers to the patch: *"I patched it by xing both test and includes usage"*.

There are also cases in which the test case failure is due to errors in the test case specification, such as flaky test. A clear example is reported in the PR #9921 from *ElemeFE/element* in which one comment points out the presence of flakiness, and the build outcome becomes green only after having merged a different PR entitled *Fix date-picker aky test #9923*, clearly related to that issue.

Finally, a recurring discussion is related to the way in which the developer sets the environment before running the testing phase. As an example, in the PR #822 of *Unitech/pm2*, a comment highlights a wrong set-up causing a failure *"pm2 kill doesn t peaceful stop the Stream Pipe, it will be destroyed until there have no WritableStream in Pipes"*, as well as a possible fix for it.

Unsurprisingly, test case failures are the ones mainly discussed in the PR review process, especially when those are related to bugs in third-party components that are not trivial to be discovered, wrong test case specifications, and wrong clean-up strategies.

Static Analysis Checks. By analyzing the sample of PRs, we identified various kinds of discussions about problems raised by static analysis tools. These include (i) styling and formatting issues in the documentation, (ii) licensing issues, (iii) malformed commit messages, and (iv) code checks violations *e.g.,* code smells, potential vulnerabilities or bugs, and code style issues. Those problems are less frequent and less discussed than test case failures as also felt by our survey respondents. Ten out of 13 respondents, never or occasionally discuss static analysis issues in the code review process, and only 2 of them think that the discussion is often useful in order to fix the problem.

Looking at some examples, developers explicitly point out style issue even in the documentation, in order to merge a PR. See for instance PR #7666 of *etcd-io/etcd*: (*"[..] etcd does not use line-breaks on markdown docs, so please put one paragraph in one line"*).

More important, we found cases in which someone suggests skipping a rule that is breaking a build because of a static analysis tool false positive. For example, in *chartjs/Chart.js* there is PR #3016 where the build failed because of a missing null check where the variable content was, instead, checked elsewhere in the code. After a long discussion, someone argues for a better configuration of static analysis tools: *"For the future, can you specify the proper JSHint settings, please?"*. The latter is also confirmed in our survey results in which 2 participants highlight the need for reconfiguring the tool used (*e.g.,* modify the properties of the rule violated or remove it) in order to address the problem.

Instead, there are cases in which the contributor has to address the violations if she wants a chance to have her PR submission accepted in the actual code base, *e.g.,* using tools that automatically fix style issues. It is the case of PR #3157 belonging to *zeit/next.js*, where a reviewer asks the contributor to execute the command "npm run lint – –fix" before the PR review starts.

Discussions related to static analysis tools concern various sorts of problems, from documentation and code style to potential bugs and vulnerabilities. While in some cases the discussions end pointing out a false positive of the tool and suggests to better configure it, in other cases, developers might want to make the linter happy in order to have the PR merged.

Con guration issues. The configuration of CI might not be trivial in many circumstances, causing itself different kinds of build failures. Such kinds of problems are discussed quite often in the analyzed sample of PRs. Also, as reported by our survey respondents, fixing such problems require complex and effort-prone activities for reconfiguring the CI pipeline.

In order to properly change the versions of the build environment without impacting the overall build process, developers have to use the *allowed_failure* functionality provided by Travis-CI. Indeed, the PR #11110 from *angular/angular.js* implies a passed build even if there is a broken build executed under the new version of *npm* (that is expected to be broken).

A very frequent problem in using CI pipeline is related to a wrong strategy used for dependencies management, such as compatibility issues between the external library and the environment or the impossibility to recover a specific dependency needed. For instance, the PR #8738 of *freeCodeCamp/freeCodeCamp* clearly states that: *"[..] react-dom 15.1 update propagated out to npm [..] before the react 15.1 update landed, causing builds to temporarily break."*. Similar problems occur when there is the need for upgrading a library towards a new version: PR #10219 of *electron/electron* mentions *"[..]*

we should use sys_mmap instead of sys_mmap2 for arm and i386".

Quite often in our dataset PRs break due to an unsuitable choice of tool support. For example, in the discussion of the PR #11693 of *adobe/brackets* project, developers point out inconsistencies between two static analysis tools, creating confusion: (*"There would be jslint annotations listed all over the code base. Is it compatible with eslint annotations?"*).

Emulators, when needed, also represent a tricky element to be configured. The PR #2716 in *AFNetworking/AFNetworking* fails because the build process attempts to use the wrong emulator (*"the 7.0 devices are really 7.0.3 devices[..] associate the runtime version with the OS version [..] and use that value instead."*).

Finally, in order to improve the build maintainability, platform-dependent commands should be avoided or used with parsimony. PR #2084 of the *BVLC/caffe* project mentions *"The rst Travis-CI build failed because* $-std = c++0x$ *was missing [..] replace the Linux-only timing commands with the standardized platform independent ones"*.

Even if developers acknowledge benefits in using CI along with PR review, its "dark side" is represented by the difficulties arising when configuring the CI pipeline. In some cases, a wrong configuration results in build maintainability problems or additional build failures.

Process-related issues. One noticeable case of process-related issue is when a PR fails to build because of unrelated problems currently being handled in a completely different (open) PR. For instance, the PR #4148 of *google/protobuf* mentioned (*"This will cause tests to fail (intendedly), as it will succeed only after PR #4147 merged."*). In this case, as also confirmed in our survey results, the discussion is not important in order to have a PR merger. In other words, the PR owner could decide to accept the PR even in the presence of the failure or to postpone its acceptance once the other change has put in (PR #16015 in *angular/angular.js*). Other cases of process-related issues are due to maintenance activities being performed on the CI pipeline, which introduced bugs in the build scripts or were simply not completed yet. For these reasons the build failed. As reported by the survey respondents, such cases might be problematic as they require a reconfiguration of the CI pipeline.

Build failures occurring when opening a PR might not be due to the actual change one wants to merge, but rather to a ripple effect caused by a different issue.

Compilation errors are very little discussed in our PRs sample even if 5 respondents declare that they usually discuss compiler-related issues and think that the discussion helps in making the PR merged.

Compilation errors might be generic issues occurring in production or test code, but also problems due to broken dependencies or to the use of a wrong compiler version (an exhaustive treatment of compiler-related build failures is in the work of Seo *et al.* [32]). While generic problems could easily be solved through private builds [10] (as also confirmed by our survey results, 8 out of 13 respondents declare to use local build to address the problem), compiler version issues are trickier and make CI useful. One such example is discussed in the PR #3868 of *gogs/gogs* in which the contributor reports that in a specific version of Go the base64.RawURLEncoding functionality is missing and that *"The *.ini parser doesn t play nice with padded base64 encodings and the unpadded encodings were introduced with Go 1.5."*.

Most compilation errors could be solved in private build, but in some cases, such as issues due to specific compiler versions, the CI server reveals to be useful.

Build hangs. These kinds of problems occur when the CI server is not able to run the build because of issues such as connection errors, not enough space on the device, or out of memory errors. Noteworthy such failures are often due to (not reproducible) transient-errors. It is very uncommon to have PRs that generate a failure due to this kind of problems even if most of them are discussed. The latter contradicts what reported in our survey results, in which 11 out of 13 respondents never or occasionally discuss build hangs issues and think that the discussion is not useful for the PR merger. One such problem, due to too many connections to a server, is discussed in the PR #14229 of *angular/angular.js*: *"Do we have too many projects using the same Sauce key?"*.

Build prepare. This category includes all problems occurring before starting the build process, *e.g.,* due to (i) fetching data on the repository and (ii) overcoming rate limits on GitHub. While the former are relatively trivial problems rarely discussed, the latter are brought to the attention of developers because depend on the way in which the CI pipeline is configured. For example, the PR #13269 of *adobe/brackets* mentions: "GitHub API rate limit exceeded" and also explains a possible root cause: *"I ll pull upstream and push to trigger the build again. The error messages show it did not get very far into the build process."*

Since that in many cases those are related to trivial issues, almost all of our respondents never discuss them and feel that the discussion is never used to fix the problem. As expected, in addressing build-prepare issues the most used action is the one aimed to re-execute the build on the CI server.

Failures belonging to build hangs are quite uncommon, even if they are always discussed during the PR review. Failures related to the build prepare, relatively easily to cope with, are rarely discussed.

IV. Threats To Validity

Threats to *construct validity* concern the relationship between theory and observation. In our study, they can be mainly due to (i) imprecisions in data extracted to build models of $\mathbf{RQ}_1$, (ii) the limited observability we could have on the PR review process by observing the discussion on GitHub. Concerning (i), besides properly testing our scripts, we followed the detailed description of how process-related factors were extracted from van der Veen s master s thesis [34]. Concerning (ii), we complemented the analysis of discussion with the inspection of build logs. In addition, to complement our study on GitHub, we performed a survey with developers.

Threats to *internal validity* concern factors internal to our study that could have influenced our results. In **RQ**$_1$, besides analyzing build-related factors in isolation, we studied their effect together with process-related factors [34], [35]. Also, we used default configurations for machine learners. We are aware that an appropriate tuning of these configurations could improve the classification results, which therefore represent a lower bound.

In **RQ**$_2$, we limited subjectiveness in discussion classification having multiple annotators for the first half of the sample.

Threats to *conclusions validity* concern the relationship between the experimentation and the outcome. Besides **RQ**$_1$ where we used, when possible, appropriate statistical procedures and effect size measures to support our conjectures, we are aware that results of our qualitative analysis are limited to a sample of our dataset. We performed a stratified sampling to obtain it, and we used a sample size of 857 PRs.

Threats to *external validity* concern the generalization of our findings. We used a number of criteria to select the projects subject of our analysis. However, it is still possible that findings from the PR quantitative and qualitative analysis do not reflect, for instance, review and CI practices in industry. Also, the number of surveyed developers is rather small, but this is mainly due to the need for limiting ourselves to experts in code review and CI. Also we intendedly avoided targeting GitHub developers to be compliant with GitHub terms of use.

V. Related Work

This section discusses literature relevant to this work and, specifically, studies related to (i) code reviews and PRs, (ii) the relationship between the use of PRs and CI, and (iii) the analysis of build failures.

A. Studies on Code Reviews

Previous studies have investigated MCR practices. Weißgerber *et al.* [38] highlighted how smaller patches have a higher chance of being accepted, while Baysal *et al.* [3] found that the code review response time and outcome is highly dependent on non-technical factors such as the patch writer experience. The latter is also confirmed by Bosu and Carver [8] who reported how changes by core developers are reviewed faster and have more chances of being accepted compared to the ones by peripheral developers. Our study complements the ones above by considering the impact of the build outcome and by studying the discussion of build failures.

Several studies have highlighted how MCR is being used sharing and transferring knowledge, educating newcomers or building a strong community [1], [4], [7], [30]. From a different perspective, German *et al.* [14] studied the fairness (*e.g.,* allocation of outcomes, dissemination of information) of code reviewing in OpenStack. Their results highlight that, even if there are many cases in which the code review process is considered fair, there are also cases in which the lack of fairness affects developers trajectory and productivity.

Kononenko *et al.* [20] investigated the developers perception of code review quality, highlighting that the feedback provided in the review process is the most important factor influencing the review quality.

Spadini *et al.* [33] studied how MCR is applied to test code. Their results indicate how the code review environment is not capable of providing some needed information, *e.g.,* achieved code coverage, or dependencies between test and production code. Therefore, developers complement the work carried out in the code review environment with local test execution. Our work attempts to bridge this gap, focusing on different artifacts involved in software development: production and test code, but also build-related configuration files.

Finally, several works analyzed the impact of code review on software quality. Some studies highlighted that there is a positive effect of code review on software quality related to the likelihood of introducing anti-patterns [26] and bugs [2], [24].

B. Studies on Pull Requests

PR discussions are a specific kind of MCR. Gousios *et al.* [15] studied PR usage in the context of distributed software development showing that the number of PRs is increasing over time. Furthermore, they identified factors highly correlated with a PR lifetime and its merging likelihood. Such factors include the project size, the number of changed files, and the test coverage.

In a follow-up qualitative study, Gousios *et al.* [16] analyzed the factors that integrators consider in their review process aimed to accept or reject a PR. Their results show that the targeted area, the existence of test cases, and the code quality are important factors influencing the PR merger. Moreover, they found a challenge related to how to prioritize PRs to be merged. To solve the latter issue, van der Veen *et al.* [34], [35] proposed an approach aimed at prioritizing PRs named PRioritizer. As we have explained in Section II-B, PRioritizer considers 14 different factors related to the PR process. Kononenko *et al.* [21], instead, studied the PR merges in an industrial project. Their results highlighted that the merge type (merge via squashing, GitHub and cherry-picking) has a significant effect on merge time. Moreover, they quantitatively found that both PR review time and merge decision are affected by PR size, discussion, and author experience, while surveying developers they identified the presence of other important factors as PR quality and type of change.

In our study, we have built a model correlating the PR merge likelihood with a number of factors, including (i) the factors considered by van der Veen *et al.* [34], [35], and (ii) build-related factors. A detailed description of such factors is reported in Table I.

C. Relationship Between Pull Requests and Continuous Integration

Previous studies have investigated the relationship existing between the usage of PRs and CI processes. First of all, Zhao *et al.* [41] by means of mining historical data of open source projects and conducting a survey among open source developers tried to identify what is the impact of adopting process automation on other development practices accounting for commit frequency, code churn, pull requests and issues closing, and testing. Their findings highlight that CI adoption align with the "commit often" rule while the "commit small" rule is only partially followed. Moreover, the number of PRs closed remains stable after the CI adoption even if the latencies on evaluating them increase.

Vasilescu *et al.* [36], mining historical data on process metrics and outcomes in 246 open source projects hosted on GitHub, have tried to quantify the effects of process automation on productivity (*i.e.,* number of PRs merged per month) and code quality (*i.e.,* number of bugs per unit time). More specifically, by using multiple regression models, they found that after the CI adoption, the number of merged PRs submitted by core developers increase while decreasing their chance of being rejected, and also the ones submitted by external contributors have a fewer chance of being rejected. However, the availability of test suite increases the chance of identifying defects in particular in PRs submitted by outsiders, resulting in a higher rejection rate. In other words, after the introduction of process automation, PRs coming from core developers have a higher chance to be accepted than others. In terms of code quality, instead, even if the introduction of CI increases the overall number of PRs that have to be handled by core members, there is not an increase on the number of user-reported bugs. The main finding is that CI adoption increases productivity without a negative effect on code quality.

Bernardo *et al.* [6], investigated the impact the adoption of CI has on the time to release PRs (*i.e.,* both merge and delivery time). By analyzing the history of 87 projects before and after CI adoption, they found that over half of the projects deliver PRs quicker after having adopted CI. Before adopting CI, the integration effort is one of the factors impacting delivery time, whereas after introducing CI the queue rank or a PR (*i.e.,* the time a PR takes to be merged compared to other PRs) impacts release cycle the most. While both their studies and ours investigate the impact of CI on PR outcome, our focus is different. Rather than studying PR delivery time, we first investigate how the build status and the evolution of builds over the PR discussion relates with the merger, and (ii) above all, qualitatively study how the results of a build failure contribute to the discussion of a PR.

The closest related study to ours is the one by Rahman *et al.* [28]. They quantitatively studied the impact of CI on code review process by mining around 600k automated build entries. They found that passed builds encourage new code review participation, and that a higher build frequency impacts on the number of comments in a PR (considered as a proxy of the review quality). Rather than looking at how passed builds encourage reviews, we study the CI builds from an opposite perspective, *i.e.,* we specifically look at build failures and investigate whether and how different kinds of build failures are discussed over PRs, with the aim of finding a problem root cause, a solution, and, ultimately, merge the PR.

D. Studies on Build Failures

Our study is aimed at identifying whether and what type of build failures are mainly discussed during the PR review process. Previous works have investigated the build failures in industrial and open source projects. Miller [25] characterized build failures in Microsoft projects, while Seo *et al.* [32] deeply investigated the compilation errors that occur in the build process at Google. Rausch *et al.* [29], instead, analyzing open source data found that test case failures are predominant in breaking the CI process. The latter have been investigated in-depth by Beller *et al.* [5] who found that testing is the core of CI and, the highest percentage of build failures has been generated by this activities. Moreover, mining open source Java projects, Zampetti *et al.* [40] looked at build failures produced by static analysis tools.

From a different perspective, Kerzazi *et al.* [19] studied the cost of dealing with build failures by observing the development of a .Net project over 6 months. The study revealed that over 2k hours of activity are devoted to dealing with build failures, with an average of one hour per build failure. Finally, Vassallo *et al.* [37] compared the distribution of build failures between OSS and the ones from a financial organization, revealing that OSS projects exhibit mainly unit testing failures while, in industry release preparation failures occur the most.

While previous studies analyzed to what extent such failures occur in builds, and how they are solved afterwards, we mainly focus on how they are brought to the attention of participants of a PR discussion, how root causes for different kinds of build failures are identified, discussed, and solved.

VI. Conclusion

We conducted an empirical study aimed at investigating the relation between Pull Request (PR) discussions and Continuous Integration (CI) build failures. Our quantitative analysis indicates that the build status upon closing a PR discussion has a limited influence on the PR merger, even if our survey respondents felt this was often the case for them. However, respondents also pointed out cases in which a PR was merged with a failed build, *e.g.,* when the build failure was related to some irrelevant static analysis warnings, or when a new PR was in the meantime opened to cope with the failure.

Our qualitative analysis indicated that the PR discussion mainly focuses on testing and static analysis problems. However, we also found many discussions related to the CI pipeline (mis)configuration. Despite the overall benefits introduced by automated continuous builds, the study points out difficulties in properly configuring and maintaining a CI pipeline, that create maintainability issues or unnecessary build failures.

Results of our study can be used to provide developers better guidance on how CI can be used together with pull request review, and code review in general. Also, it highlights the complexity of dealing with CI in certain situations, *e.g.,* when dealing with emulated environments, non-deterministic (flaky) tests, or different environments exhibiting an inconsistent behavior.

REFERENCES

[1] A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013.

[2] G. Bavota and B. Russo, “Four eyes are better than two: On the impact of code reviews on software quality,” in *IEEE International Conference on Software Maintenance and Evolution, (ICSME)*, 2015.

[3] O. Baysal, O. Kononenko, R. Holmes, and M. Godfrey, “The influence of non-technical factors on code review,” in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, 2013.

[4] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, “Modern code reviews in open-source projects: Which problems do they fix?” in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014.

[5] M. Beller, G. Gousios, and A. Zaidman, “Oops, my tests broke the build: An explorative analysis of travis ci with github,” in *Proceedings of the International Conference on Mining Software Repositories (MSR)*. ACM, 2017.

[6] J. H. Bernardo, D. A. da Costa, and U. Kulesza, “Studying the impact of adopting continuous integration on the delivery time of pull requests,” in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, 2018.

[7] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, “Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft,” *IEEE Transactions on Software Engineering*, 2017.

[8] A. Bosu and J. C. Carver, “Characteristics of the vulnerable code changes identified through peer code review,” in *36th International Conference on Software Engineering, ICSE 14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*.

[9] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Lawrence Earlbaum Associates, 1988.

[10] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*. Addison-Wesley Professional, 2007.

[11] M. E. Fagan, “Design and code inspections to reduce errors in program development,” *IBM Systems Journal*, 1976.

[12] R. A. Fisher, “Confidence limits for a cross-product ratio,” *Australian Journal of Statistics*, 1962.

[13] K. Gallaba, C. Macho, M. Pinzger, and S. McIntosh, “Noise and heterogeneity in historical build data: an empirical study of travis CI,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*.

[14] D. M. Germán, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, “"was my contribution fairly reviewed?": a framework to study the perception of fairness in modern code reviews,” in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*.

[15] G. Gousios, M. Pinzger, and A. van Deursen, “An exploratory study of the pull-based software development model,” in *36th International Conference on Software Engineering, ICSE 14, Hyderabad, India - May 31 - June 07, 2014*.

[16] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen, “Work practices and challenges in pull-based development: The integrator s perspective,” in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*.

[17] M. A. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update,” *SIGKDD Explorations*, 2009.

[18] F. E. Harrell Jr, with contributions from Charles Dupont, and many others., *Hmisc: Harrell Miscellaneous*, 2017, r package version 4.0-3. [Online]. Available: https://CRAN.R-project.org/package=Hmisc

[19] N. Kerzazi, F. Khomh, and B. Adams, “Why do automated builds break? an empirical study,” in *30th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014.

[20] O. Kononenko, O. Baysal, and M. W. Godfrey, “Code review quality: How developers see it,” in *Proceedings of the 38th International Conference on Software Engineering*, 2016.

[21] O. Kononenko, T. Rose, O. Baysal, M. Godfrey, D. Theisen, and B. de Water, “Studying pull request merges: a case study of shopify s active merchant,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 2018.

[22] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, “Understanding variable importances in forests of randomized trees,” in *Proceedings of the 26th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2013.

[23] B. W. Matthews, “Comparison of the predicted and observed secondary structure of t4 phage lysozyme,” *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 1975.

[24] 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 *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.

[25] A. Miller, “A hundred days of continuous integration,” in *Proceedings of the Agile 2008*, 2008.

[26] 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. of the 22nd Int l Conf. on Software Analysis, Evolution, and Reengineering (SANER)*, 2015.

[27] S. Panichella, V. Arnaoudova, M. Di Penta, and G. Antoniol, “Would static analysis tools help developers with code reviews?” in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015.

[28] M. M. Rahman and C. K. Roy, “Impact of continuous integration on code reviews,” in *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*.

[29] T. Rausch, W. Hummer, P. Leitner, and S. Schulte, “An empirical analysis of build failures in the continuous integration workflows of java-based open-source software,” in *Proceedings of the 14th International Conference on Mining Software Repositories*. ACM, 2017.

[30] P. C. Rigby, D. M. German, and M.-A. Storey, “Open source software peer review practices: A case study of the apache server,” in *Proceedings of the 30th International Conference on Software Engineering*, 2008.

[31] P. C. Rigby and M.-A. Storey, “Understanding broadcast based peer review on open source software projects,” in *Proceedings of the 33rd International Conference on Software Engineering*, 2011.

[32] H. Seo, C. Sadowski, S. G. Elbaum, E. Aftandilian, and R. W. Bowdidge, “Programmers build errors: a case study (at Google),” in *Proc. Int l Conf on Software Engineering (ICSE)*, 2014.

[33] D. Spadini, M. F. Aniche, M. D. Storey, M. Bruntink, and A. Bacchelli, “When testing meets code review: why and how developers review tests,” in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*.

[34] E. van der Veen, “Prioritizing pull requests,” Delft, The Netherlands, 2014.

[35] E. van der Veen, G. Gousios, and A. Zaidman, “Automatically prioritizing pull requests,” in *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*.

[36] B. Vasilescu, Y. Yu, H. Wang, P. T. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in github,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*.

[37] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella, “A tale of CI build failures: An open source and a financial organization perspective,” in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*.

[38] P. Weißgerber, D. Neu, and S. Diehl, “Small patches get in!” in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 2008.

[39] F. Zampetti, G. Bavota, G. Canfora, and M. Di Penta, “A study on the interplay between pull request review and continuous integration builds. Replication Package https://goo.gl/KjTpxp.”

[40] F. Zampetti, S. Scalabrino, R. Oliveto, M. Di Penta, and G. Canfora, “How open source projects use static code analysis tools in continuous integration pipelines,” in *Proceedings of the 14th International Conference on Mining Software Repositories*. ACM, 2017.

[41] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, “The impact of continuous integration on other software development practices: a large-scale empirical study,” in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*.