2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

Insights into Continuous Integration Build Failures

Md Rakibul Islam
University of New Orleans, USA
Email: mislam3@uno.edu

Minhaz F. Zibran
University of New Orleans, USA
Email: zibran@cs.uno.edu

***Abstract*—Continuous integration is prevalently used in modern software engineering to build software systems automatically. Broken builds hinder developers' work and delay project progress. We must identify the factors causing build failures.**

This paper presents a large empirical study to identify the factors such as, complexity of a task, build strategy and contribution models (i.e., push and pull request), and projects level attributes (i.e., sizes of projects and teams), which potentially have impacts on the build results. We have studied 3.6 million builds over 1,090 open-source projects. The derived results add to our understanding of the role of those factors on build results, which can be used in minimizing build failures.

I. Introduction

Continuous integration (CI) systems provide facilities for automatic compilation, building, testing and deployment of a software [7] usually triggered by a unit of changes committed to a larger code base. Since its inception in 1991 as one of the twelve Extreme Programming (XP) practices [4], CI has become a widely accepted practice in software development community [5]. Although CI is continuously gaining popularity in software engineering, it has received very little attention from the research community [7], [6]. Despite having few quantitative studies [7], [9], [10], [11] on CI, research community lack quantifiable evidence on the implications of adoption and use of CI [6].

While the usage of CI improve the productivity and quality of software products [11], broken builds halt the development work of the entire team for a significant amount of time [8]. Thus, we must identify the factors that cause build failures. If we can identify those factors, developers can take cautious measurements to minimize their impacts and thus can substantially reduce development time. In this work, we quantitatively examine various development factors that cause broken builds. In particular, we address the following three research questions.

RQ1: *Is there any relationship between the complexity of a task and* CI *build failure i.e., broken build?*

— We hypothesize that a task with higher complexity increases the likelihood of a build failure. Kerzazi et al. [8] reported that probability of a build failure increases with the increase in number of changed code lines and number of changed files (that are parts of the factors to measure complexity of a task) included in a push initiating the build. We will verify their findings by conducting an in-depth analysis using a larger dataset.

RQ2: *Do the build strategies, and the developers' contribution models have any impact on* CI *build failures*?

— A build strategy (i.e., choosing a build tool to run a build at a specific time in a particular build branch) can have impacts on the build results. For example, developers typically take more care when writing changes to a *master* branch than to a *non-master* branch, which can be related to higher number of successful builds in the master branch [7].

Again, different features (e.g., readability and simplicity of the build configuration files) offered by the different build tools may play role in build results (e.g., successful or broken). In the similar way, developers contribution models such as, *direct push* and *pull request* can be attributed to build results. Here, we want to examine all those causal relationships using quantitative analyses. Findings from our examination can help a development manager to set his build strategy and to choose a suitable contribution model for a project to prevent build failures.

RQ3: *Do the sizes of teams and projects have any correlation with* CI *build failure*?

— Dependencies among the team members and even among code components can be increased when the sizes of the team and the project are large. Such dependencies, particularly in code, are susceptible to broken builds [9]. Identifying any correlation between the build failures and the sizes of teams and projects will be helpful for developers to be more cautious to prevent build failures when those sizes become large.

II. Data Collection

To address the aforementioned research questions, we collect the snapshot revision *travistorrent_11_1_2017.csv.gz* of a large dataset prepared by Beller et al. [6]. The collected dataset (i.e., snapshot) consists of source code and builds' information of 1,300 software projects developed in Java and Ruby.

All those information are collected from three different sources (1) *Travis CI*- a CI tool (2) *GIT*- a version control system and (3) *GitHub*- a collaboration platform. Then, those collected information are combined together to prepare the dataset. For example, for a particular build, the dataset consists of the attributes such as, *build ID*, *build status/result*, *build duration* and *build tool*, which are collected from the *Travis CI*. Then, those attributes are combined with other attributes such as, *number of changed code lines and changed files*, *commit ID* and *number of commits* of that build, which are collected from *GIT* and *GitHub*. Further details about the dataset can be found elsewhere [6].

To ensure a project has used CI service sufficiently, we select only those projects from the aforementioned dataset,

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.30

which have meet the two criteria– (1) projects using a CI service for at least one year, (2) projects having at least 100 builds regardless of their success or failure status. In this way, we select 1,090 software projects consists of 3.6 million builds for the study.

Categorization of Builds Results: In the final dataset we find five types of build results such as *passed, failed, errored, canceled* and *start*. We exclude all the builds, which have *start* status, as their final results are unknown [5]. We consider the build result *passed* as *successful* and the remaining build results are termed as *unsuccessful* throughout the paper.

III. Analysis and Findings

The research questions RQ1, RQ2 and RQ3 are respectively addressed in Section III-A, Section III-B and in Section III-C. To verify the statistical significance of the results derived from our quantitative analysis, we also apply the statistical *Mann-Whitney-Wilcoxon (MWW)* test [3] with α = 0.05 for RQ1 and RQ3. We use *Chi-squared* [12] test of independence with the same value of α for RQ2. To measure the effect sizes, we use *Cohen's d* [1] value and *Cramer's V* [2] value along with *MWW* test and *Chi-squared* test, respectively.

A. Analysis of Complexity of a Task

We consider the complexity of a task is higher if the number of code churns, the number of changed files and the number of built commits in a single push for that task are higher. In our investigation, we use both source code churn and test code churn separately that will give us deeper level insights of impacts of code churn on build results.

Source Code Churn (SCC). SCC is defined as the number of changed lines i.e., lines added, deleted and modified to the files in a build. We investigate whether the number of SCC in a single build have any relationship with *unsuccessful* build. To do that for each of the selected projects, we compute the average number of SCC per build in both *successful* and *unsuccessful* builds. The box-plots in Figure 1a present the distributions of the computed averages of SCC per build in logarithmic scale for each of the projects in each type of the build results (i.e., *successful* and *unsuccessful*). The 'x' mark indicates the average number of SCC per build over all the projects. We notice from Figure 1a that both the median and average of SCC over all the projects are higher in the *unsuccessful* builds compared to those in the *successful* builds.

To determine the statistical significance of the observation, we conduct a *MWW* test between the distributions of average scores of SCC in each type of the build results for all the projects. The computed P-value ($P = 2.74 \times 10^{-15}, P < \alpha$) reveals that the difference is statistically significant. Moreover, the *Cohen's d* test returns the value 0.34, which suggests that the effect size is medium between the distributions of the average scores of SCC in each type of the build results for all the projects.

Test Code Churn (TCC). Again, we compute the average number of TCC per build in both *successful* and *unsuccessful* builds for each of the projects. The box-plots in Figure 1b present the distributions of computed average scores of TCC per build for each of the projects in both *successful* and *unsuccessful* builds. Again, from Figure 1b we see that both the median and average of TCC over all the projects are higher in *unsuccessful* builds compared to those in *successful* builds.

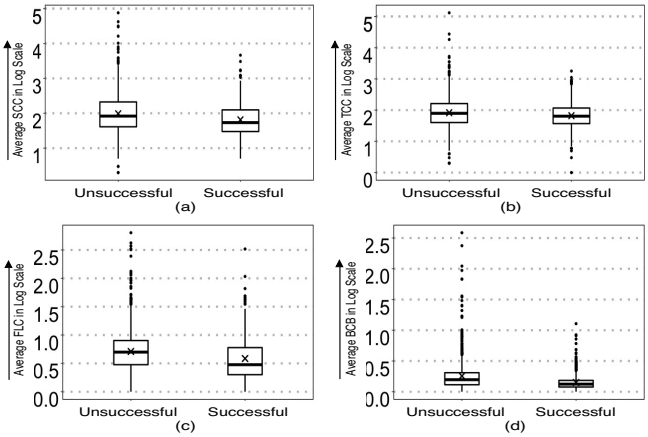

Fig. 1. Distributions of the average scores of (a) number of source code churns (SCC), (b) number of test code churns (TCC), (c) number of changed files (FLC) and (d) number of built commits (BCB) according to build results in projects.

To measure the statistical significance, we conduct a *MWW* test between the distributions of averages of TCC in *successful* and *unsuccessful* builds for all the projects. The resulted P-value ($P = 3.31 \times 10^{-7}, P < \alpha$) implies that the difference is statistically significant, however, the computed *Cohen's d* value 0.213 indicates a small effect size.

File Level Change (FLC). We calculate the number of changed files i.e., FLC by summing up the number of files added, deleted and modified in a push or pull request to a development branch that initiates the build. Then, we compute the average number of FLC per build in both *successful* and *unsuccessful* builds for all the projects and plot those average scores in Figure 1c. We observe from Figure 1c that both the median and average of FLC over all the projects are higher in *unsuccessful* builds compared to those in *successful* builds. The computed P-value ($P = 2.2 \times 10^{-16}, P < \alpha$) of a *MWW* test between the distributions of average scores of FLC in both *successful* and *unsuccessful* builds for all the projects implies the difference is significant. Moreover, the resulted value 0.36 of *Cohen's d* test suggests a medium effect size.

Built Commits in a Build (BCB). To identify the impact of number of BCB on build results, we compute the average number of BCB in both *successful* and *unsuccessful* builds for each of the projects. The box-plots in Figure 1d present the computed average scores of BCB for each of the projects in both *successful* and *unsuccessful* builds. Again, we see from Figure 1d that both the median and average score of BCB over all the projects are higher in *unsuccessful* builds compared to those in *successful* builds.

The computed P-value ($P = 2.2 \times 10^{-16}, P < \alpha$) of a *MWW* test between the distributions of average scores of BCB in both *successful* and *unsuccessful* builds for each of the projects, indicates that the difference is statistically significant.

TABLE I
BUILDS RESULTS USING DIFFERENT TYPES OF BUILD TOOLS

Build Tool	Frequency of Builds Result		Proportion of Build Result	
	Successful	Unsuccessful	Successful	Unsuccessful
Plain	47,459	70,649	40.19%	59.81%
Ant	296,534	221,940	57.20%	42.80%
Maven	182,977	124,895	59.44%	40.56%
Ruby	1,857,234	748,924	71.27%	28.73%
Gradle	55,800	17,088	76.56%	23.44%

TABLE II
BUILD RESULTS IN DIFFERENT BUILD BRANCHES

Build Branch	Frequency of Build Result		Proportion of Build Result	
	Successful	Unsuccessful	Successful	Unsuccessful
Master	1,806,181	838,977	68.28%	31.72%
Non-master	633,823	344,519	64.78%	35.22%

Moreover, the computed *Cohen's d* value 0.61 suggests that the effect size is large.

Based on the observations and statistical analyses, we derive the answer to the RQ1 as follows:

Ans. to RQ1: *Complexity of a task i.e., the number of built commits, the number of source code churns and the number of changed files in that task have a statistically significant relationship with* unsuccessful *builds.*

B. Analysis of Build Strategy and Contribution Model

In this study we consider a build strategy is comprised of taking decisions about selecting two components of a build- (1) build tools such as, *Ant*, *Maven*, *Gradle*, *Ruby*, and *Plain* and (2) build branches i.e., *master branch* and all other development branches termed as *non-master branch*. Again, based on the ways developers commit their changes in the development branches, we consider there are two types of contribution models- (1) *direct push* and (2) *pull request*. While direct push is the most prevalent contribution model, pull-request is gaining significant popularity in open-source projects. Here, we examine the impacts of *build tools*, *build branches* and *contribution models* on build results.

Types of Build Tools (TBT). Table I presents the frequencies of builds according to their results found in different types of build tools. To gain a deeper picture, we calculate the proportion of builds results for each build tool as shown in the right most two columns in Table I. The tool *Gradle* shows the highest proportion of *successful* builds followed by the tool *Ruby* (i.e., *rake*). The tools *Ant* and *Maven* show almost equal proportions of *successful* builds. While for every tool, proportion of *successful* builds is higher compared to proportion of *unsuccessful* builds, interestingly, exception can be observed when *Plain* i.e., shell or other languages' scripts are used to run the builds. The same tool also shows the highest proportion of *unsuccessful* builds followed by the tools *Ant* and *Maven*. A *Chi-squared* test (χ^2 = 93680, df = 4, $P = 2.2 \times 10^{-16}, P < \alpha$) also indicates statistical significance of the relationship between uses of build tools and build results with a medium effect size (as *Cramer's V* = 0.168).

Types of Development Branches (TDB). Table II presents the frequencies and proportions of builds according to their build results found in both *master* and *non-master* branches of development. We see from Table II that both the frequency and the proportion of *successful* builds are higher in the *master* branch compared to the non-master branch. A *Chi-squared* test (χ^2 = 3971.14, df = 1, $P = 2.2 \times 10^{-16}, P < \alpha$) indicates statistical significance of the relationship between the development branches and build results, although the effect size is weak (as *Cramer's V* = 0.0331).

TABLE III
BUILD RESULTS CATEGORIZED IN ACCORDANCE WITH DIFFERENT CONTRIBUTION MODELS

Build Model	Frequency of Build Result		Proportion of Build Result	
	Successful	Unsuccessful	Successful	Unsuccessful
Push	2,044,644	1,009,142	66.96%	33.04%
Pull request	395,360	174,354	69.40%	30.60%

Types of Development Model (TDM). Table III presents the frequencies and proportions of builds according to their results found in both *direct push* and *pull request* models of contribution. Although *push* model has higher number of *successful* builds, the proportion of *successful* builds is higher in *pull request* model. A *Chi-squared* test (χ^2 = 1301.5, df = 1, $P = 2.2 \times 10^{-16}, P < \alpha$) indicates a statistical significance of the relationship between the development models and the build results with a weak effect size (as *Cramer's V* = 0.019). Based on the observations and the statistical analyses, we derive the answer to the RQ2 as follows:

Ans. to RQ2: *A build tool has a statistically significant impact on the probability of the build results.*

C. Analysis of Project Level Attributes

We examine whether the sizes of project level attributes such as, the source code size in terms of lines of code (SLOC) and the test code size per 1,000 SLOC, have any correlation with the build results. We also check the sizes of the teams - measured in terms of the number of contributors in that project - to relate with the build results.

Size of Source Code (SSC). We calculate the average number of SSC per build in *successful* and *unsuccessful* builds for each of the projects. The distributions of those calculated averages are presented in Figure 2a where the median and the average score over all the projects are found almost equal in the *successful* and *unsuccessful* builds. The computed P-value ($P = 0.881, P > \alpha$) of a *MWW* test between the average scores of SSC in the *successful* and *unsuccessful* builds for all the projects also implies that the difference is not statistically significant.

Size of Test Code (STC). Similar to SSC, for each of the projects we compute the average number of STC per build in *successful* and *unsuccessful* builds. The box-plots in Figure 2b present the distributions of computed averages of STC per build for each of the projects in *successful* and *unsuccessful* builds. The computed P-value ($P = 0.708, P > \alpha$) of a *MWW* test indicates no significant difference between averages of STC in *successful* and *unsuccessful* builds.

Team Size of a Project (TSP). Again, for each of the projects we compute the average number of TSP per build in both *successful* and *unsuccessful* builds. The box-plots in Figure 2c present the distributions of the computed averages of TSP per build for each of the projects in *successful* and *unsuccessful* builds. The computed P-value ($P = 0.529, P > \alpha$) of a *MWW* test indicates no statistical significant difference

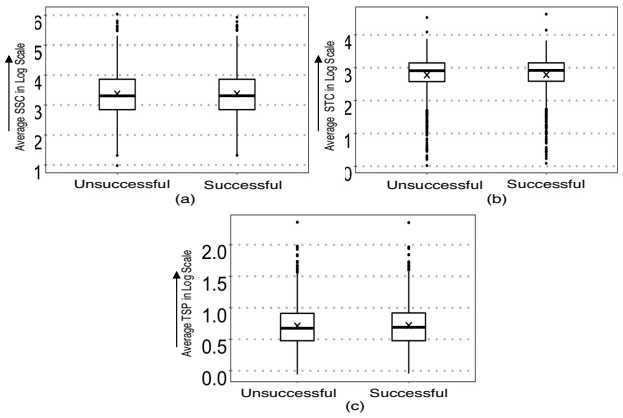

Fig. 2. Distributions of the averages of (a) size of source code (SSC), (b) size of test code (STC), (c) team size (TSP) according to build results in projects.

between averages of TSP in *successful* and *unsuccessful* builds. Based on the analyses, we derive the answer to the RQ3 as follows:

Ans. to RQ3: *Sizes of projects and teams have no correlation with build results.*

IV. Threats to Validity

Exclusion of 193 projects from the dataset can be questioned. One may also question the exclusion of builds with *start* status. Despite of all such exclusions, our studied dataset consists of 1,090 projects and 36.2 million builds, which are significantly large numbers for our quantitative analysis. We consider a single push that initiates a build represents a task, which may not be always true as a task can be comprised of multiple pushes.

All the studied projects are developed in either Java or Ruby, so the generalizability of the findings can be considered as a threat to validity of the study. The methodology of data collection, analysis, and results are well documented in this paper. Hence, it should be possible to reproduce the study results.

V. Related Work

Beller et al. [5] identified that testing is the single most important reason why builds fail. While their work mainly focused on impacts of tests on build failures, we have considered many other important factors, which have statistically significant relationships with build results.

The work of Kerzazi et al. [8] is the most relevant to our work where they examined the impacts of the number of code churns, number of changed files and sizes of teams on build results in addition to a qualitative study. While both the works agree on the negative impacts of higher number of code churns and changed files on builds results, contradictory results can be observed on the relationship between teams' sizes and *unsuccessful* builds. We gain more confidence on our results as we have conducted our study on 1,090 projects and 36.2 million builds, while the former study was based on only one project and 3,214 builds.

Vasilescu et al. [10] identified higher number of *successful* builds in *pull request* model than in *direct push* model (by using 223 GitHub projects), although we have not found any statistically significance difference in the number of *successful* and *unsuccessful* builds between *direct push* and *pull request* models using a larger dataset.

Hilton et al. [7] conducted a study to identify some costs and benefits (e.g., productivity) of projects using CI. Vasilescu et al. [11] also examined the productivity and quality (measured in terms of detecting bugs early before releases) of projects that use CI. Both the works found positive impacts of CI on productivity, while later work claimed that the increased productivity comes without negative effect on quality of a product. Instead of measuring the effects of using CI, we have examined what factors cause *unsuccessful* builds.

VI. Conclusion

In this paper, we have presented a large-scale quantitative empirical study on the impacts of various development factors on build results. We have studied 3.6 million builds over 1,090 open-source projects.

In our study, we have found that build results are significantly affected by the of number of changed lines of code, number of changed files, and number of built commits in tasks. We have also identified correlation between build tools and build results. However, the number of changed test code lines, development branches and contribution models have no significant impacts on the results of builds.

The findings from this work are validated in the light of statistical significance. The results from this study substantially advance our understanding of the impacts of development factors on the build results, although contradictory results (as discussed in Section V) indicate the need for further investigations along those directions. In future, we also plan to conduct more studies on the impacts of using CI in projects.

References

[1] *Cohen's d*. http://trendingsideways.com/index.php/cohens-d-formula/, last access: Feb 2017.

[2] *Cramer's V*. http://www.real-statistics.com/chi-square-and-f-distributions/effect-size-chi-square/, last access: Feb 2017.

[3] D. Anderson, D. Sweeney, and T. Williams. *Statistics for Business and Economics*. Thomson Higher Education, 10th edition, 2009.

[4] K. Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[5] M. Beller, G. Gousios, and A. Zaidman. Oops, my tests broke the build: An analysis of travis ci builds with github. In *PeerJ Preprints*, 2016.

[6] M. Beller, G. Gousios, and A. Zaidman. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *MSR*, 2017.

[7] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *ASE*, pages 426–436, 2016.

[8] N. Kerzazi, F. Khomh, and B. Adams. Why do automated builds break? An empirical study. In *ICSME*, pages 41–50, 2014.

[9] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge. Programmers' build errors: A case study (at Google). In *ICSE*, pages 724–734, 2014.

[10] B. Vasilescu, S. Schuylenburg, J. Wulms, and M. Brand A. Serebrenik. Continuous integration in a social-coding world: Empirical evidence from github. In *ICSME*, pages 401–405, 2014.

[11] B. Vasilescu, Y. Yu, H. Wang, P Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in github. In *ESEC/FSE*, pages 805–816, 2015.

[12] M. Zibran. *CHI-Squared Test of Independence*. https://pdfs.semanticscholar.org/0822/f125a21cfbd05e5e980c8017499fb966568f.pdf, last access: Feb 2017.