Travis, Why Do the Builds Keep Failing?: An Analysis on CI Failure Causes and Severity

Colin Cummings
Computer Science 496
University of Nebraska-Lincoln
Lincoln, NE
ccummings3@huskers.unl.edu

Easton Joachimsen
Computer Science 496
University of Nebraska-Lincoln
Lincoln, NE
ejoachimsen@huskers.unl.edu

Matt Sichterman
Computer Science 496
University of Nebraska-Lincoln
Lincoln, NE
mattsichterman@huskers.unl.edu

***Abstract*—Continuous Integration is a practice that has become widely used and is a standard in the industry. Travis CI can be dropped into any GitHub repository and automate various tasks including the build and test processes. With that much convenience at face value, there also comes hiccups when builds fail which can get tedious. This paper strives to quantify the different types and reasons that builds fail and how quickly a successful subsequent build is completed. Using TravisTorrent's data set [1], we apply various data analysis and data visualization techniques to clearly compare and portray failing builds to understand their cause and severity based on the time to the next successful build.**

***Index Terms*—continuous integration, continuous delivery, build failures, Travis CI, GitHub**

I. Introduction

Finding the relationship between CI/CD and what kinds of problems are being caught in the pipelines are important to determine the complexity of bugs being caught. We plan to utilize the TravisTorrent data set [1] and continue to research past work to help us determine what types of problems are most often causing pipeline failures while considering time needed to fix these problems and uncover specific data to show what most often is causing failures so that developers can have a better idea of what to look out for.

Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub, *How Open Source Projects use Static Code Analysis Tools in Continuous Integration Pipelines*, and *How Does Contributors' Involvement Influence the Build Status of an Open-Source Software Project?* are all papers that analyze the Travis-Torrent data-set that tracks multiple different areas on how users interact with Travis-CI through the analysis of builds and how they pass or fail. Each of these papers seems to refine the overarching data-set into extremely low level patterns such as test cases, static code analyzers, and user experience. However, none of these papers address the most frequent reason for build failures or even utilize half of the data-sets true potential.

Despite the other related studies surrounding CI/CD including how unit tests impact the way that CI is integrated, contributors' involvement and the effect on the build process, and the effect of static code analysis in continuous integration pipelines, there is still work needed to determine what is causing failures most often and how complex the problems found are with respect to the time to the next successful build after a failure.

We aim at answering the following primary research question:

> What types of problems are causing continuous integration pipelines to fail most often?

To answer this question, we performed a quantitative analysis on the TravisTorrent data set [1] to better understand what kinds of problems are most commonly caught in the CI/CD pipelines, with an emphasis on the average time it took to recover from identified build failures.

II. Related Work

CI has become an integral quality assurance practice [2], but what exactly has sparked this interest in CI and how is it used? Well in software engineering, continuous integration (CI) is a practice in which contributions from multiple developers are integrated back into a shared mainline multiple times every day, additionally through the use of this process the quality of their contributions is also able to be monitored. There have been a great amount of studies performed collecting and analyzing different aspects of CI answering a great number of unknowns about the practice and providing insight into its usages.

The paper *Oops, My Test Broke the Build - An Explorative Analysis of Travis CI with GitHub*, authors Moritz Beller, Georgios Gousios, and Andy Zaidman provide further insights into the importance of tests and their contributions to how CI is used. Within their work they provide a purely quantitative analysis on the impact and usage of tests within Travis CI. The research was derived from the TravisTorrent data set provided by GitHub and limited their analysis window to projects that were written with Ruby and Java. When examining the results to their tests they state that testing is the single most important reason for integration to break, more prevalent than compile errors, missing dependencies, build cancellations and provisioning problems together [3].

Authors Renato O. Santo, Gustavo Pinto, and Fernando Castor explore the impacts of a subjects involvement within a project to the CI method in their paper *How Does Contributors' Involvement Influence the Build Status of an Open-Source Software Project?*. They explore data within the TravisTorrent data set to try to find a pattern that suggests that non-casual developers are more likely to introduce code to Travis CI that will cause the build to fail. Within their analysis they combine both the data from the data set along with data extracted from the Travis API in order to gather statistics of users, their contribution percentages, and the amount of builds with their respective status (pass or fail). Their findings suggest that there is that in eighty five percent of the cases, there is no representative difference between contributions placed from casual and non-casual contributors, meaning that being a casual is not strong indicator for creating failing builds [5].

Another technical paper that we took influence from was *How Open Source Projects use Static Code Analysis Tools in Continuous Integration Pipelines* written by Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. Their goal was to further investigate static code analysis and its role in the continuous integration practices. Their investigation efforts were divided into three primary questions: which tools are being used and how they are configured for the CI, what types of issues make the build fail or raise warnings, and whether, how, and after how long are broken builds and warnings resolved [6]. Upon conducting their analysis they found that there is a huge variability in terms of how strict the projects had their analysis tools set to. Certain tools were more strict than others and were meant to accomplish different tasks. In the end this was found to be a key point when analyzing data, the separation of tools used to interrupt builds and the tools used to warn about possible bugs or vulnerabilities. Results of these findings greatly impacted the remaining questions of the paper in the way that the second and third research questions were very dependant on the first. That being said the findings for *RQ2* are very similar to the findings of *RQ1*. As for *RQ3* they found that the time taken to resolve broken builds caused by static code analyzers was almost immediately fixed (in median within 8 hours), in most cases by actually fixing the problem in the source code, and only in few cases by modifying the build script [6].

III. Study Design

A. Study Goal

Following our main research question presented in the introduction, we defined the main goal of our quantitative analysis by applying the Goal Question Metrics template as so:
Analyze the TravisTorrent data set
for the purpose of evaluating pipeline failures
with respect to the type of errors causing the failures and the time elapsed until the next successful build
from the point of view of the developer
in context of Travis CI data mined and linked back to GitHub

B. Research Questions

With this, we defined and explored research questions as follows:

RQ1. Do failing unit tests cause the most failures in continuous integration pipelines?
RQ2. Does the majority (i.e., greater than 50%) of continuous pipeline failures result from issues that take less than 1 hour to resolve?
RQ3. Does the size of a commit or the number of jobs in a pipeline correlate with pipeline failures more frequently?

C. Data Set

The TravisTorrent data set contains a deep analysis of the project source code, process and dependency status of 1,359 projects. The data set comes filtered in an effort to only include projects that that are non-forks, non-toys, have greater than 10 watchers on GitHub, and have a Travis CI history of greater than 50 builds. In addition, 936 of the projects in the data set use Ruby and 423 projects use Java. Some notable projects in the data set include Ruby on Rails, Google Guava and Guice, Chef, RSpec, Checkstyle, ACIIDoctor, Ruby and Travis [4].

TravisTorrent provides very convenient access to the data set, which can be queried directly using Google Big Query or managed and manipulated locally using a SQL dump download or CSV file. We opted to use each of these techniques in an attempt to maximize our productivity during the study.

The data provides us with an extensive amount of columns giving us an abundance of information about the Travis CI data and associated GitHub source code data. Some of the columns of most importance to our study include gh_project_name, git_prev_commit_resolution_status, git_diff_src_churn, gh_diff_src_files, tr_status, and tr_log_bool_tests_failed [1] among others. More information can be seen on the TravisTorrent website.

D. Hypotheses

Based on the related work that we reviewed and personal experience that we have accumulated in the space of continuous integration, we expect unit test failures to most commonly cause the pipelines to fail. Therefore, we formulated the following null hypothesis to check the frequency of unit tests being the cause of failures in the Travis CI pipelines:

- $H0_{FREQ}$: The frequency in which unit tests cause continuous integration pipelines to fail is greater than the frequency of all other single causes of failure.

We formulated the following null hypothesis to check the relative severity (i.e., time taken until next successful build) of issues that cause continuous pipeline failures:

- $H0_{SEV}$: The severity of issues caught in continuous integration pipelines is low (i.e., take less than 1 hour to resolve) a majority of the time.

Finally, we formulated the following null hypothesis to check which factor correlates with build failures more frequently:

- $H0_{COR}$: The size of a commit correlates with overall pipeline failures more frequently than just the number of jobs within a pipeline.

The alternative hypothesis were formulated as follows:

- $H1_{FREQ}$: The frequency in which unit tests cause continuous integration pipelines to fail is no greater than the frequency of any other single cause of failure.
- $H1_{SEV}$: The severity of issues caught in continuous integration pipelines is low (i.e., take less than 1 hour to resolve) no more than 50% of the time.
- $H1_{COR}$: The size of a commit correlates with overall pipeline failures no more frequently than just the number of jobs within a pipeline.

We formulated $H0_{FREQ}, H0_{SEV}$, and $H0_{COR}$ to study RQ1, RQ2 and RQ3, respectively.

E. Methodology

Since we used a pre-established data set, our study focused mainly on quantitative analysis. First, we imported the TravisTorrent data set into a Jupyter notebook via Google Colab in the form of a CSV file. Google Colab gives us the ability to run our analysis in the cloud with free, dedicated GPUs while utilizing Python as our programming language of choice. In order to focus our attention on the most important properties, we filtered the data set quite a bit to only include what we saw as most vital to our results. To do so, we used the Python package called ”pandas” to help us better organize and aggregate the data.

After organizing the data, we decided to follow a fairly structured process to analyze the data in three main steps, following our research questions closely:

Step 1: Identify the cases in which tr_status equals ”failed” and determine the cause of the failure. To do so, we considered metrics such as if tests failed, the number of tests failed, and the overall number of tests. This analysis allowed us to easily visualize the cause of failures and identify the frequency of failing pipelines with respect to test failures and other factors.

Step 2: Identify the cases in which the Travis CI pipeline ”failed” and determine the elapsed time until the next successful build. To do so, we considered factors such as tr_status (the build status), tr_prev_build which provides information about the previous build, gh_pushed_at (timestamp of the push that triggered the build provided by GitHub), gh_build_started_at (timestamp of the push that triggered the build provided by Travis CI), and more.

Step 3: Compare the factors associated with each build failure in an effort to find if the number of jobs or size of commit correlates with pipeline failures more frequently. In order to do so, we used visualization techniques and clustering in order to identify a correlation in the mentioned factors to pipeline failures.

Much of our analysis was done manually. We used the ”matplotlib” library to generate visualizations in order to better show trends in the data. In order to help with our analysis, we also utilized techniques such as clustering in attempts to try and automate the identification processes in all 3 steps. We utilized the ”scikit-learn” library in Python to do this.

IV. Analysis and Results

A. Descriptive Statistics

W.I.P.

B. Statistical Inference

RQ1: W.I.P.
RQ2: W.I.P.
RQ3: W.I.P.

C. Additional Results

W.I.P.

V. Discussion

A. Answers to the Research Questions

B. Threats to Validity

Internal Validity: Our experience in not only the topic of continuous integration, but also our experience in conducting software research as a whole can potentially have an impact on the validity of this report. The selection threat of our data sets plays an important role in the results yielded as the characteristics of the data can potentially fluctuate depending on the filtering of different attributes which decide the projects to be included or excluded from the data analyzed. To deal with this threat we have decided to use the approach of prioritizing data set quality rather than quantity.

Conclusion Validity: W.I.P.

External Validity: We analyzed data sets specific to projects that used GitHub and Travis CI. Therefore, it may pose a threat to validity when generalizing the results for projects that use something different than Travis CI for their continuous integration. The projects being analyzed were likely all completing different tasks and functionalities, this can lead to a threat of interaction of setting and treatment, since various projects may have inconsistencies in their usage and dependability on continuous integration within each project.

C. Implications and Future Extensions

D. Lessons Learned

VI. Conclusions and Future Work

A. Summary of Findings

There are still a number of questions surrounding the usage statistics of Travis CI. Through our finding we have answered... (WIP)

B. Future Work

Additional research into the topics of CI is still needed and questions like ... have yet to be answered... (WIP)

References

[1] The Test Roots Team, ”Access TravisTorrent: Free and Open Travis Analytics for Everyone”, 2020.

[2] P. M. Duvall, S. Matyas, and A. Glover, Continuous integration: improving software quality and reducing risk. Pearson Education, 2007.

[3] M. Beller, G. Gousios and A. Zaidman, ”Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub,” 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 356-367, doi: 10.1109/MSR.2017.62..

[4] M. Beller, G. Gousios and A. Zaidman, ”TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration,” 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 447-450, doi: 10.1109/MSR.2017.24.

[5] M. Rebouças, R. O. Santos, G. Pinto and F. Castor, ”How Does Contributors’ Involvement Influence the Build Status of an Open-Source Software Project?,” 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 475-478, doi: 10.1109/MSR.2017.32.

[6] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora and M. Di Penta, ”How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines,” 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 334-344, doi: 10.1109/MSR.2017.2.