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

An Empirical Study of the Personnel Overhead of Continuous Integration

Marco Manglaviti, Eduardo Coronado-Montoya, Keheliya Gallaba, and Shane McIntosh
Department of Electrical and Computer Engineering
McGill University
Montréal, Canada
{marco.manglaviti,eduardo.coronado-montoya,keheliya.gallaba}@mail.mcgill.ca, shane.mcintosh@mcgill.ca

***Abstract*—Continuous Integration (CI) is a software development practice where changes to the codebase are compiled and automatically checked for software quality issues. Like any software artifact (e.g., production code, build specifications), CI systems require an investment of development resources in order to keep them running smoothly.**

In this paper, we examine the human resources that are associated with developing and maintaining CI systems. Through the analysis of 1,279 GitHub repositories that adopt Travis CI (a popular CI service provider), we observe that: (i) there are 0 to 6 unique contributors to CI-related development in any 30-day period, regardless of project size; and (ii) the total number of CI developers has an upper bound of 15 for 99.2% of the studied projects, regardless of overall team size. These results indicate that service-based CI systems only require a small proportion of the development team to contribute. These costs are almost certainly outweighed by the reported benefits of CI (e.g., team communication and time-to-market for new content).

I. Introduction

Continuous Integration (CI) is a common practice in the software industry to integrate source code changes into the official repositories of software projects. Under CI, each incremental change to a software system is automatically compiled (if necessary), tested, and scanned for quality issues. In theory, CI improves defect reporting and reduces developer overhead, since unlike less frequent build cycles, problems can be caught early, while tradeoffs and design decisions are still fresh in the developers' minds.

CI is particularly useful for software projects where a project code base and/or development team is large and distributed. Maintaining stability in such a code base becomes a particularly challenging issue, especially as systems tend to accrue complexity as they age [1]. Open source projects often need to deal with issues of large and distributed teams, since these projects often undergo many changes that are submitted by a variety of contributors.

Travis CI is one of the most popular cloud-based platforms for providing CI services to software projects. The use of Travis CI in a software project requires the presence of a configuration file, namely `.travis.yml`, in the root directory of its repository. This file specifies the programming language runtimes, their versions and other environment parameters for building and testing the project. It is also used to define all CI-related tasks that are to be executed during the integration process, the order in which these tasks must to be executed, and the external tools that are needed. Therefore, modifying this configuration file relates to time spent by developers on the development and maintenance of the CI specification.

As with any software technology, adopting Travis CI comes at a cost—developers must spend time and effort on the development and maintenance of the CI specification. For example, CI specifications need to change in order to adapt to changing source code requirements and dependencies.

In this paper, we study the personnel overhead that is introduced by adopting Travis CI in open source software projects. More specifically, we examine the trend between the growth of Travis CI contributors as team sizes increase, and projects age. Through the analysis of the MSR Challenge dataset [2], we address the following research questions:

(RQ1) How quickly do Travis CI teams grow?
We find that the Travis CI team size tends to plateau. Indeed, Travis CI team size is consistently in the range of 0–6 developers per 30-day time period for 99.9% of the studied projects.

(RQ2) Is there a relationship between the Travis CI team size and the total team size?
No, despite observing project team sizes of up to 917 contributors, we do not find any Travis CI teams larger than 41 contributors.

These results lead us to conclude that the personnel overhead that is introduced by Travis CI is not directly proportional to both the team size and development time. Specifically, the results demonstrate that with an increasing contributor base and an aging software system, the overall personnel cost of maintaining CI tends to stabilize.

Paper organization. The remainder of the paper is organized as follows. Section II describes the design of our case study, while Section III presents the results. Finally, Section IV draws conclusions and outlines avenues for future work.

II. Case Study Design

In this section, we outline the two-step approach that we use to analyze the MSR Challenge dataset [2] in order to address our research questions. Figure 1 provides an overview of our approach. Below, we describe each step in the approach.

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.31

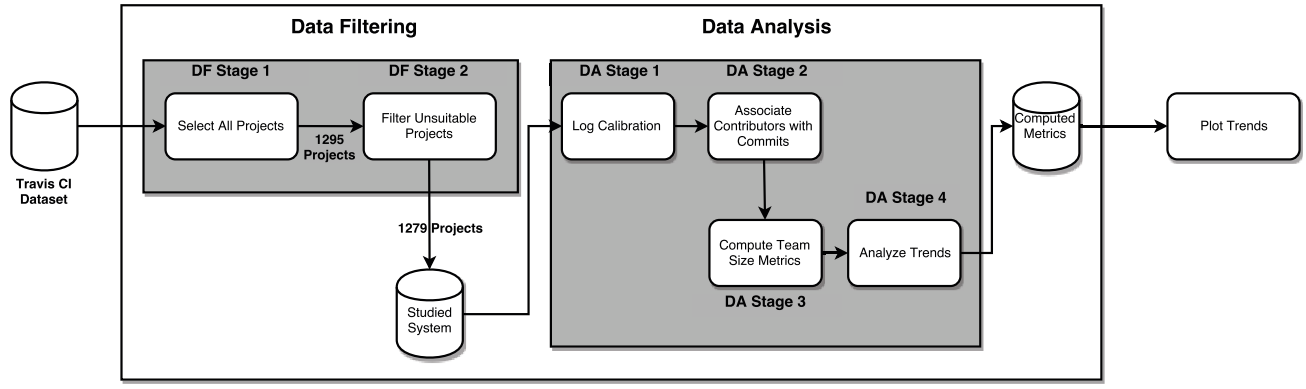

Fig. 1. An overview of our approach to study the MSR Challenge data set.

A. Data Filtering

The challenge data set [2] identifies 1,300 GITHUB repositories using TRAVIS CI to manage their build data. The challenge data set was used to identify projects with at least 50 TRAVIS CI builds and minimum 10 watchers on GITHUB [2]. However, not all of these repositories were available or suitable for our analysis. Thus, we first filter the set of repositories according to the steps that we outline below.

DF 1: Filter unavailable projects. We began our study by selecting all of the projects in the TRAVIS CI data set [2]. Next, we attempted to clone the corresponding GITHUB repositories for each project. Of the 1,300 projects that were mentioned in the TRAVIS CI data set, 1,295 could be cloned from GITHUB at data collection time (October 3rd, 2016). The five projects that were no longer available for analysis included four deleted projects and one project that had switched from GITHUB's open-source plan to the private one.

DF 2: Filter unsuitable projects. Software development is a rapidly moving target. Technologies like TRAVIS CI are adopted and abandoned at a breakneck pace. To prevent projects that are no longer using TRAVIS CI from introducing noise in our analyses, we filter out projects that no longer contain a `.travis.yml` file in their repositories. Of the 1,295 projects that we selected in Stage DF 1, 1,279 repositories still contained a `.travis.yml` file at data collection time.

B. Data Analysis

After filtering the raw dataset, 1,279 repositories survive for further analysis. Prior to conducting our analysis, we normalize the repository data to a common timespan (DA 1), associate contributor IDs with each commit (DA 2), compute team size metrics (DA 3), and analyze trends in team size (DA 4).

DA 1: Log Calibration. In order to produce a more precise analysis of the trend of contributors to the TRAVIS CI specifications over time, we opt to align the commit logs of each project at the first commit to the `.travis.yml` file. This allows us to compare the number of (and trend in) project and CI contributors across the projects with a common starting point. The timespan of analysis for all projects was truncated at the data collection date, i.e., since each repository for a project in the studied system was cloned locally for use in the data analysis phase, the commit logs for each project terminate at the last commit prior to October 3rd, 2016. Since each studied project may have adopted TRAVIS CI at different times, the studied projects have different time spans. Figure 2 provides a density plot of the duration of the studied projects. Moreover, Figure 3 shows the concentration of projects with commit activity in a given time period using the shade of the hexagon-shaped bins.

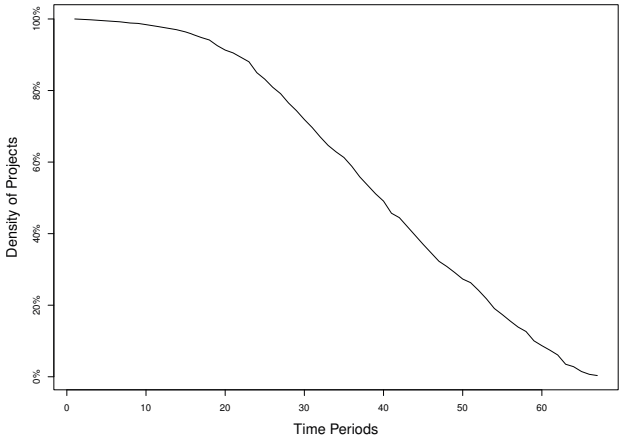

Fig. 2. The percentage of the projects with commit activity in each time period.

DA 2: Associate contributors with commits. For each studied repository, we iterate over every commit in the studied time span in order to identify contributors. We use the author email field in the metadata of the commit as a contributor ID.

Past work has noted that email addresses in open-source data sets are noisy [3]. For example, two (or more) email addresses may refer to the same individual. In order to disambiguate email aliases that refer to the same person, we use a similar approach to that of Bird et al [3]. We remove the domain from the email address and normalize the name by removing all

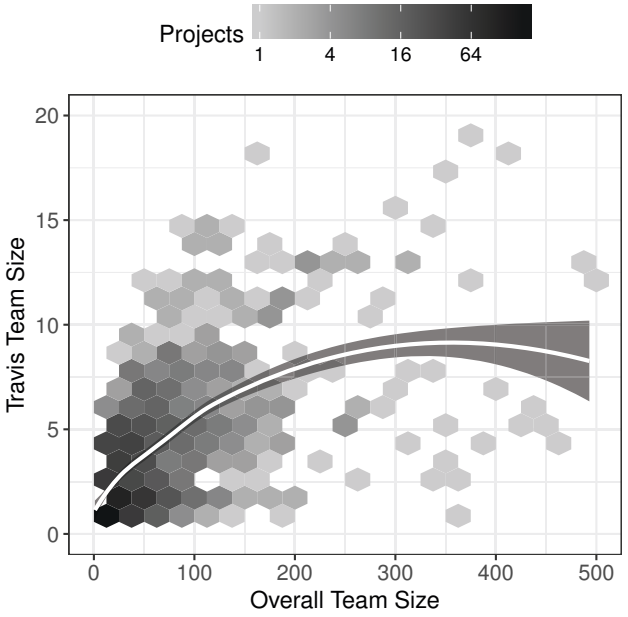

Fig. 4. TRAVIS CI team size plotted against overall team size. The white line indicates a Loess-smoothed curve with a shaded area indicating the 95% confidence interval.

punctutation. We then measure the Levenshtein edit distance between the name and any previous ID for the project in question. If the distance is within a threshold of 3, we attribute both email addresses as being the same contributor.

DA 3: Compute team size metrics. Now that each commit has been assigned to a contributor, we compute two types of contributor metrics. First, to serve as a baseline, we compute the number of unique contributors to all of the repository files, i.e., the magnitude of the set of unique contributor IDs who have made contributions to the repository in question. Next, we compute the number of unique contributors to the TRAVIS CI specifications, i.e., the magnitude of the set of unique contributor IDs for the `.travis.yml` file.

DA 4: Analyze trends. Finally, to address our research questions, we analyze our two contribution metrics in two ways. First, to address RQ1, we analyze our contribution metrics with respect to time. We use 30-day periods to recompute our contribution metrics and study their trends over time. Second, to address RQ2, for each studied system, we compute one overall measurement of our contribution metrics using the entire time span. Then, we plot the TRAVIS CI team size measurements against the overall team size measurement.

III. CASE STUDY RESULTS

In this section, we discuss the results of our case study with respect to our two research questions. For each research question, we first present our approach to addressing it and then discuss the results that we observe.

RQ1: How quickly do TRAVIS CI *teams grow?*

Approach. To address RQ1, we compute the TRAVIS CI and overall team sizes for each studied project in 30-day time periods. Figure 3 plots the results of each project using scatter plots. To address concerns of overplotting (i.e., several points appearing in a small space), we apply hexagonal binning to the raw scatter plot data. The darker the shade of any given hexagon, the more projects that fall within the bin.

Results. As projects age, the TRAVIS CI team size tends to plateau. Figure 3 shows that as the studied projects age, the number of unique contributors to the `.travis.yml` file remains relatively constant. On the other hand, the overall team size shows a more quickly and consistently growing trend. This suggests that as projects accumulate contributors, the proportion of the team that needs to maintain the TRAVIS CI specifications does not need to grow proportionally.

Focusing on the left quadrant of Figure 3 (i.e., Project Team Size), we observe that the projects in the data set have a wide range of growth rates after the TRAVIS CI configuration file has been introduced (recall that time period 0 indicates when the TRAVIS CI configuration file is introduced). Indeed, there are projects like Spree that grow rapidly, growing from a team size of 203 to 695. Most other projects grow more slowly.

On the other hand, the right quadrant of Figure 3 (i.e., TRAVIS CI Team Size) demonstrates that the team that maintains the `.travis.yml` file does not exceed 6. Note that having 0 contributors to the `.travis.yml` file indicates the file was not modified as part of any commit during that 30-day time period. This is particularly meaningful because it demonstrates that as projects grow and an increasing number of contributors are modifying source code, the number of developers modifying the TRAVIS CI configuration file tends to plateau.

> ***Observation 1****: The number of contributors to the* TRAVIS CI *specification is consistently between 0 and 6 developers per 30-day time period over a project's lifetime*

(RQ2) Is there a relationship between the number of CI contributors and the total team size?

Approach. We compute the total number of contributors to the `.travis.yml` file and the overall team size of each project. Figure 4 shows the scatter plot for these two values in all of the studied projects, with hexagonal binning used to address overplotting. In addition to the hexagon-binned scatter plot, Figure 4 shows a trend line using a Loess-smoothed regression (white line) with a 95% confidence interval (translucent black band). For clarity, we have removed one project with extreme values (1000+ contributors) from the dataset when plotting Figure 4. The shade of each hexagon is proportional to the number of projects that fall within the bin.

Results. When we set out to explore this research question, our initial intuition was that when a project becomes popular and attracts contributors, the number of contributors who need to modify the `.travis.yml` file would increase respectively. As can be observed in Figure 4, The total number of TRAVIS CI contributors remains relatively constant—fewer than 15

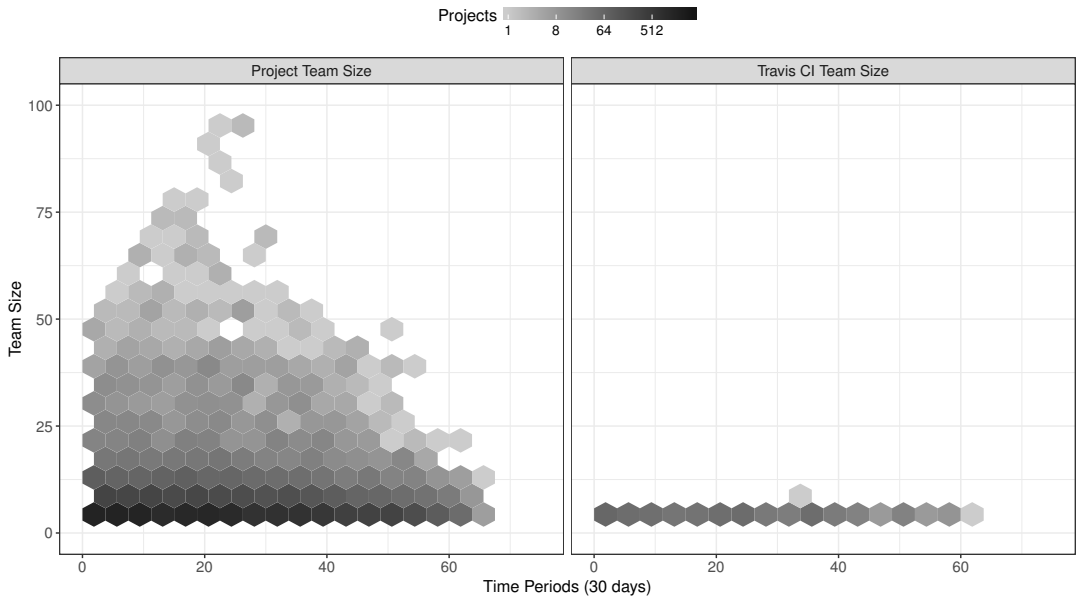

Fig. 3. TRAVIS CI and overall team size per 30-day time period.

developers contribute to the `.travis.yml` file of 100% of the studied projects with a total team of 50 or fewer developers. The trend increases slightly with the growth of a project's team size, but is not proportional to team size.

By focusing on the range of x-axis (total project contributors) as compared to the range of the y-axis (total TRAVIS CI contributors), we observe that projects with many total contributors still have roughly the same number of TRAVIS CI contributors as projects with few total contributors. Indeed, while there is a slight increase in the number of contributors to the `.travis.yml` file, the increase is minimal and plateaus between 10-15 total contributors for projects with 50 contributors or more. This is emphasized by the Loess-smoothed regression line in Figure 4. With the exception of a few projects, all of the projects with more than 100 contributors have a total number of TRAVIS CI contributors of 15, with the exception of 7 projects. This contradicts our initial intuition that the number of contributors to the TRAVIS CI configuration file would increase proportionally as team size increases. This would, in theory, allow teams to scale up while dedicating a relatively small team to CI maintenance and configuration. Future work could examine the trend between the advantages of CI (i.e. increased successful builds) and the personnel overhead associated with maintaining CI.

These results lead us to conclude that the personnel overhead of CI for open source software projects tends to decrease with project growth, since only a smaller percentage of developers make changes to the TRAVIS CI configuration file as the team size increases.

***Observation 2**: The number of contributors to the* TRAVIS CI *configuration file is bounded at 15 in almost all cases regardless of team size.*

IV. CONCLUSIONS

Adopting new software development practices and technologies usually come at the cost of the initial development, maintenance, and operation of these new technologies. In this paper, we observe that:

- As projects age, the maintenance costs of TRAVIS CI plateaus within the range of 0 to 6 contributors during any given 30-day time period, regardless of team size.
- As projects accumulate contributors, the number of TRAVIS CI contributors remains relatively consistent at a maximum of 15 developers.

These results lead us to conclude that for projects with growing contributor bases, adopting CI becomes increasingly beneficial and sustainable as projects age. Indeed, maintenance effort of the TRAVIS CI configuration file does not in fact increase over time. Instead, we show that over an increasing project development lifecycle, CI maintenance and development is unaffected by team growth. In future work, we plan to investigate commit size and content of TRAVIS CI contributions.

In order to aid in future replication of our results, we make our data and scripts publicly available online.[1]

REFERENCES

[1] L. A. Belady and M. M. Lehman. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

[2] M. Beller, G. Gousios, and A. Zaidman. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *Proceedings of the 14th Int'l Conf. on Mining Software Repositories (MSR)*, 2017.

[3] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining Email Social Networks. In *Proceedings of the 3rd Int'l Conf. on Mining Software Repositories (MSR)*, pages 137–143, 2006.

[1] `https://dx.doi.org/10.6084/m9.figshare.4803172`