On the rise and fall of CI services in GitHub

Mehdi Golzadeh
Software Engineering Lab
University of Mons
Mons, Belgium
mehdi.golzadeh@umons.ac.be

Alexandre Decan
Software Engineering Lab
University of Mons
Mons, Belgium
alexandre.decan@umons.ac.be

Tom Mens
Software Engineering Lab
University of Mons
Mons, Belgium
tom.mens@umons.ac.be

***Abstract*—Continuous integration (CI) services are used in collaborative open source projects to automate parts of the development workflow. Such services have been in widespread use for over a decade, with new CIs being introduced over the years, sometimes overtaking other CIs in popularity. We conducted a longitudinal empirical study over a period of nine years, aiming to better understand this rapidly evolving CI landscape. By analysing the development history of 91,810 GitHub repositories of active npm packages having used at least one CI service, we quantitatively studied the evolution of seven popular CIs, specifically focusing on their co-usage and migration in the considered repositories. We provide statistical evidence of the rise of GitHub Actions, that has become the dominant CI service in less than 18 months time. This coincides with the fall of Travis that has seen an important decrease in usage, likely due to a combination of policy changes and migrations to GitHub Actions.**

***Index Terms*—Continuous integration, distributed software development, software repositories, GitHub**

I. Introduction

Continuous integration (CI), deployment and delivery have become the cornerstone of collaborative software development and DevOps practices. CI automates the integration of code changes from multiple contributors into a central repository where automated builds, tests and code quality checks run. Well-known examples of CI services are Jenkins, Travis, CircleCI and AppVeyor. CI services can also be built-in in social coding platforms such as GitHub and GitLab [1]. GitLab already featured CI capabilities since November 2012. Based on popular demand, and in response to CI support integrated in GitLab, GitHub publicly announced the beta version of GitHub Actions (abbreviated to GHA in the remainder of this article) in October 2018. In August 2019, they officially began supporting Continuous Integration through GHA, and the product was released publicly in November 2019.

GHA [2] allows to automate a wide range of tasks based on a variety of triggers such as commits, issues, pull requests, comments and many more. GHA can be used to facilitate code reviews, code quality analysis, communication, dependency and security monitoring and management, testing, etc. GHA facilitates the integration with external services, and can even obviate the need of using such external services altogether.

GitHub is by far the largest social coding platform, hosting the development history of millions of collaborative software repositories, and accommodating over 56 million users in September 2020 [3]. Given its popularity and the ease with which GHA allows to automate the CI workflow, we hypothesise that GHA has had a significant impact on today's CI landscape. More particularly, we believe that it has increased the awareness of the need for CI, it has reduced the entry barrier for projects to start using CI, and it may have lead projects to migrate from other CI services towards GHA.

This article aims to quantitatively and objectively verify these hypotheses, and discusses their consequences, through a longitudinal analysis of how different CIs have been used over a nine-year period in 91,810 GitHub repositories corresponding to the software development history of reusable Node.JS packages distributed through the npm package registry. This empirical study focuses on four research questions:

RQ_1 *How did the CI landscape evolve?* We identified 20 different CIs being used in the considered set of repositories, some of which were considerably more prevalent than others. Together with Travis, GHA covers more than 80% of all usages. Moreover, in only 18 months GHA has overtaken all other CIs in popularity.

RQ_2 *What are the most frequent combinations of CIs?* We observed that many repositories have used multiple CIs during their lifetime. AppVeyor is nearly always used in combination with some other CI. If a repository uses a CI simultaneously with another one, it is mostly in combination with Travis, GHA or CircleCI.

RQ_3 *How frequently are CIs being replaced by an alternative?* We observed a non-negligible amount of CI migrations. GHA attracted most of these migrations. The majority of migrations were moving away from Travis and towards GHA.

RQ_4 *How has the CI landscape changed since GHA was introduced?* Based on a regression discontinuity design, we found that the usage of Travis, Azure and CircleCI has been negatively affected by the introduction of GHA.

This article is structured as follows. Section II motivates the selected dataset and discusses the data extraction and cleaning steps that were carried out. Sections III to VI provide answers to each research question. Section VII discusses the ramifications of these answers. Section VIII presents the threats to validity of the conducted research. Section IX presents the related work. Finally, Section X concludes.

II. Data Extraction

In order to analyse the use of CIs in software development repositories on GitHub, we need a large dataset containing

TABLE I: Most popular CI services, number and (cumulative) proportion of repositories using them, and (cumulative) proportion of usages, in decreasing order.

			repositories			**usages**	
CI	**URL**	**first observed on**	**#**	**%**	**cum. %**	**%**	**cum. %**
Travis	http://travis-ci.org	Jun 10, 2011	53,401	58.2%	58.2%	44.9%	44.9%
GHA	http://github.com/features/actions	Jan 23, 2019	46,416	50.6%	90.9%	39.0%	83.9%
CircleCI	http://circleci.com	Jan 15, 2014	11,431	12.4%	98.1%	9.6%	93.5%
AppVeyor	http://www.appveyor.com	Apr 04, 2014	3,553	3.9%	98.3%	3.0%	96.5%
Azure	http://azure.microsoft.com	Sep 11, 2018	1,045	1.1%	98.7%	0.9%	97.3%
GitLab CI	http://docs.gitlab.com/ee/ci	Sep 02, 2015	1,018	1.1%	99.1%	0.9%	98.2%
Jenkins	http://www.jenkins.io	Mar 30, 2016	1,008	1.1%	99.6%	0.8%	99.0%
Others	N/A	Oct 23, 2013	1,138	1.2%	100.0%	1.0%	100.0%

thousands of GitHub repositories for a wide range of software programming projects serving different purposes and exhibiting variations in longevity and size. The dataset should exclude repositories that have been created merely for experimental or personal reasons, or that only show sporadic traces of commit activity [4]. Registries of reusable software packages (e.g., npm for JavaScript or Maven for Java) are good candidates to find such large datasets, as they typically host thousands of software packages at different levels of maturity and popularity. However, not all packages belonging to such registries have an associated git repository on GitHub.

According to the libraries.io open-source monitoring service, npm is by far the largest of all listed package registries [5]. We used the public API of npm to list all 1.6M+ scoped packages. We downloaded the metadata of each package on 23 May 2021 and found that 803K packages mention an associated git repository hosted on GitHub. We cloned 676K of these repositories, the remaining ones corresponding to repositories that were no longer available. Since one of our goals is to study how the CI landscape has evolved these recent years, we excluded repositories that were not *active* during the last year of the observation period (i.e., they had no commit between 24 May 2020 and 23 May 2021). We also excluded 11,557 forks, since part of their history duplicates the one of the forked repository. This left us with 201,403 repositories.

CI usage in a repository is typically visible through the presence of some specific CI-related configuration files. For example, the presence of a .travis.y(a)ml file indicates that Travis CI has been configured for this repository while a .y(a)ml file in the .github/workflow/ folder triggers GHA. Based on an exploration of scientific publications, blog posts and developer websites, we established an initial list of 28 candidate CI tools and services to consider. We carefully went through their documentation to determine the file paths that must be considered to detect the presence of each CI. We excluded CIs that are mostly configured through a dedicated UI and not a configuration file, or CIs whose configuration file cannot be detected (e.g., because the file path and file name can be freely chosen by the users). The threats related to this approach are discussed in Section VIII. After such exclusion 20 CIs remained: Travis, GHA, CircleCI, AppVeyor, Azure, Jenkins, GitLab CI, Drone CI, Hound CI, Bitbucket CI, Wercker, Golang CI, CodeBuild, Buildkite, Semaphore, Cirrus, CloudBees, Amplify, Buddy and Bitrise.

We then checked every commit of all 201,403 cloned repositories for the presence of CI-related configuration files. We found 179,535 distinct CI-related file paths in 95,035 repositories. We relied on these file paths as a proxy to detect if and when a CI was used by a repository.

Working with git histories can be quite challenging [6] and leads to a range of data quality issues that need to be dealt with. A first issue is that git is revisionist, allowing one to rewrite the history of a repository. Unfortunately, there is no way to detect such rewritings, except when it leads to inconsistencies (e.g., commits referring to files that were not yet created or that were already removed, commits with invalid dates, etc.). We identified and removed 60 repositories for which we found such inconsistencies for CI-related files. We contacted the maintainers of two of such repositories, who confirmed the inconsistencies, explaining that it was the consequence of an earlier migration to git and a merge of different repositories into a single one.

Another issue stems from the presence of implicit branches and the fact that the history of a git repository is represented by a directed acyclic graph, as opposed to a tree-like structure in svn. As a result, chronological sequences of commits may originate from distinct branches, and may contain *a priori* contradictory changes (e.g., a file that is added and removed multiple times over relatively short periods of time). Since we determine the use of a CI based on the presence of specific file paths, we are particularly exposed to this issue. We found several cases of chronological sequences of CI-related files being added and removed multiple times in a row. Since such file removals did not correspond to a deliberate intent to remove the corresponding CI service, we ignored all removals of CI-related files for which the same file was found to be reintroduced within 30 days. We empirically found that this value allows to capture 98.1% of the cases we found, while preserving actual cases of re-introductions of a CI.

After this data cleaning step, we used the presence of CI-related files to determine when a CI was added and (possibly) removed from a project. One should note that a same CI can be added and removed more than once during a project's lifetime. By manually inspecting CI usages in repositories, we noticed several cases of repositories experimenting with the integration of a CI for a few days only. We excluded such cases by removing 6,910 usages (found in 6,586 repositories) whose duration did not exceed 30 days. We also found and excluded 3

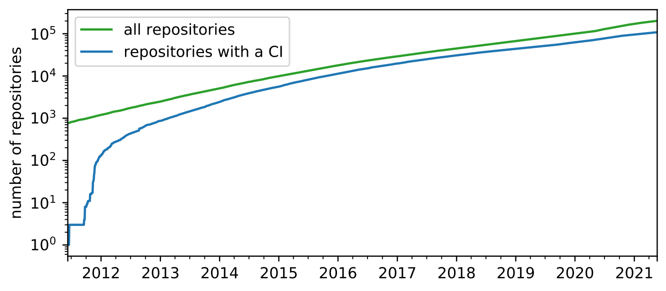

Fig. 1: Evolution of the number of repositories (green line) and number of repositories using a CI (blue line).

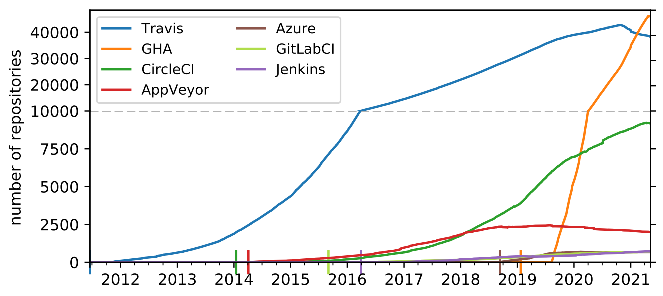

Fig. 2: Number of repositories using a specific CI.

cases of repositories making use of a high number of different CIs at the same time. We manually inspected these repositories and found that their purpose was to showcase integration of an app with various CIs. An example of such a repository is cypress-io/cypress-example-kitchensink.

The final dataset contains 119,033 CI usages spread across 91,810 repositories. The higher number of usages than repositories signals that there are many repositories that use multiple CIs. Table I provides an overview of the CIs found in repositories, along with the number of usages and the number of repositories in which they were found at least once through time. Travis and GHA are by far the most popular CIs, being used each by more than half of the repositories with a CI and together covering more than 90% of all repositories with a CI. This may be expected for Travis, since it has existed since 2011 and thus repositories have had more time to use this service. It is surprising to find GHA as the secondmost popular CI, given that its first usage is observed in 2019 only, making it the most recent of the considered CIs. The five next popular CIs, following at quite a distance, are CircleCI, AppVeyor, Azure, GitLab CI and Jenkins. The remainder of this article exclusively focuses on these seven CIs since, together, they represent 99% of all CI usages, and cover 99.6% (91,426) of all repositories having used a CI.

The data and code to replicate the analysis in this article are available on https://doi.org/10.5281/zenodo.5815352.

III. RQ_1: How did the CI landscape evolve?

RQ_1 is exploratory in nature, aiming to obtain a better understanding of which CIs are found in the repositories of our dataset, how prevalent these CIs are, and how this has been changing over time. This will provide the necessary context to interpret the results of the subsequent research questions.

Obviously, not all repositories make use of a CI. Fig. 1 shows the evolution of the number of repositories in our dataset that use a CI compared against the total number of repositories we considered at that time. We observe that the number of repositories using a CI has a tendency to increase over time. Proportionally to the number of considered repositories, it went from 56.7% in January 2015 to 63% in January 2020. At the end of the observation period (i.e., in May 2021), the proportion of repositories with a CI is 53.6%. This lower proportion is likely to be a consequence of the fact that recent repositories did not have enough time yet to adopt a CI. According to Hilton et al. [7], the median time to adopt a CI is one year.

More than half of all repositories in the dataset are using a CI, suggesting that CI usage is prevalent and has become an inseparable aspect of software development for many projects.

These observations are in line with the ones of Hilton et al. [7], and confirm that CIs are widely used in practice nowadays. However, that does not mean that all CIs are equally used. Fig. 2 breaks down the blue curve of Fig. 1 for the seven most popular CIs, showing the evolution of the number of repositories using each CI. Since the number of repositories using Travis and GHA has a different order of magnitude than for the other CIs, we used a different y-axis scale starting from 10,000 repositories (illustrated by the horizontal dashed line).

We observe that the oldest CI (Travis) has dominated the landscape since its introduction in 2011. This is not surprising since Travis was the only available CI for the first two years of the observation period (at least in our dataset). This explains why Travis accounts for the highest number and proportion of repositories in Table I. Other popular CIs have entered the CI landscape only more recently: CircleCI and AppVeyor emerged in 2014, GitLab CI in 2015, and Jenkins in 2018. Since their introduction in the first half of 2014, CircleCI and AppVeyor were successful in attracting developers and obtained a good share of repositories using these CIs. Even GitLab CI, originally introduced as an integrated CI service for GitLab, started to attract a small share of GitHub repositories since September 2015. Although Jenkins was one of the earlier CI tools for Java applications, it only shows up in our data as of April 2016, and it has a small share of repositories using it. This can be explained by two phenomena. First of all, our dataset focuses on npm packages, hence they are less likely to contain Java development. Secondly, the *pipelines-as-code* feature[1] only became available in Jenkins in April 2016, implying that we were only able to track repositories using this feature as of that date. Finally, we found evidence of a rapid growth of repositories using Azure following its introduction in 2018, but after a short period of time the

[1]https://developers.redhat.com/blog/2016/08/24/whats-new-in-jenkins-2-0-2

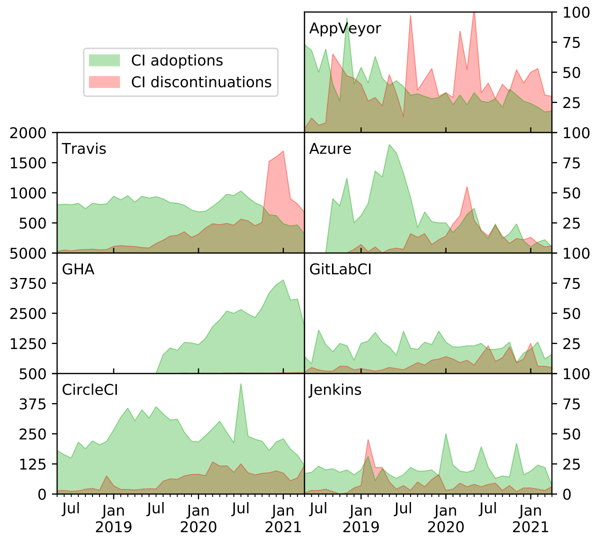

Fig. 3: Monthly number of repositories adopting and discontinuing a CI.

number of repositories using Azure stagnated.

Probably the most surprising phenomenon that can be observed in Fig. 2 is the very rapid growth of GHA usage. Despite the fact that GHA was only introduced in 2019, it has become the second most popular CI in our dataset just 18 months after its introduction, even overtaking the dominant position of Travis! At the end of the observation period, GHA covers 51.7% of the repositories with a CI, followed by Travis (42.5%), CircleCI (10.2%) and AppVeyor (2.2%). The remaining CIs together (Azure, GitLab CI and Jenkins) do not exceed 2.3%. This suggests that GHA has changed the CI landscape dramatically, fuelling the need to carry out a deeper empirical analysis in the remainder of this article.

Together, at the end of the observation period, Travis and GHA dominate the CI landscape, as they are used by 90.1% of all repositories with a CI. It took only 18 months for GHA to overtake Travis in popularity.

From Fig. 2, we observed for most CIs either a slight recent decrease in the number of repositories using it (Travis and AppVeyor) or a reduced growth in this number (CircleCI, Azure and GitLab CI). These observations can be the consequence of less repositories adopting the CIs, of more repositories discontinuing them, or a combination of both. Fig. 3 shows the monthly churn in CI usage for the last three years of the observation period, computed as the monthly number of repositories that adopted and discontinued each CI.

We observe that the recent decrease in the number of repositories using Travis is due to a combination of less repositories adopting Travis (from July 2020 onwards) and of many repositories that discontinued using them. This is especially visible from late 2020 onwards, where the monthly number of repositories that discontinued using Travis went from 508 in October to 1,520 in November 2020.

We also observe that the reduced growth of CircleCI, AppVeyor and Azure is mostly a consequence of less repositories adopting these CIs. For instance, the number of monthly adoptions of CircleCI went from an annual average of 308 in 2019 to 253 in 2020, and only 173 in 2021. Similarly, the number of adoptions of Azure dropped from 90 in May 2019 to only 17 in February 2020, reaching the number of discontinuations. In AppVeyor, there are even more discontinuations than adoptions since July 2019.

Interestingly, most of the churn we observed for these CIs started a few months after the introduction of GHA. GHA is the only considered CI that exhibits a steadily increasing adoption rate, and nearly no discontinuations (only 374 compared to 46,416 adoptions during the observation period). We will therefore analyse and discuss the impact of GHA on the CI landscape in detail in RQ_4.

The adoption rate of Travis, CircleCI, AppVeyor and Azure decreased. The discontinuation rate increased for Travis, Azure and GitLab CI. GHA has a steadily increasing adoption rate and very little discontinuations.

IV. $RQ2$: What are the most frequent combinations of CIs?

Table I revealed that there are more CI usages than repositories. This implies that some repositories use more than one CI (either simultaneously or at different points in time). Table II shows the number of repositories in function of the number of distinct CIs they have used during their observed lifetime. While three out of four repositories have only used a single CI throughout their lifetime, there is still a large proportion that have used 2 different CIs (21.7%) or even more than two (3.3%). This confirms that it is not unusual for a repository to use multiple CIs throughout its lifetime.

One out of four repositories having used a CI has used at least two different CIs during its observed lifetime.

Fig. 4 shows the proportion of repositories having used a given pair of CIs A and B. Since we found that most repositories having used multiple CIs involved Travis (86.2%), GHA (83.7%) and CircleCI (22.3%), and to avoid the analysis to be biased by these most frequently used CIs, we computed the proportions relative to the number of repositories having used a given CI (this is, the proportion of repositories having used CIs A and B relative to all repositories having used A). For example, 30.5% of the repositories with Travis have used GHA, 5.5% have used AppVeyor and 4.8% CircleCI. Note that the sum may exceed 100% (e.g., for AppVeyor) since repositories may have used more than two CIs.

TABLE II: Number and proportion of repositories having used different CIs during their lifetime.

# CIs →	**1**	**2**	**3**	**4**	**5**
# repos.	68,549	19,799	2,647	371	60
% repos.	75.0%	21.7%	2.9%	0.4%	0.07%

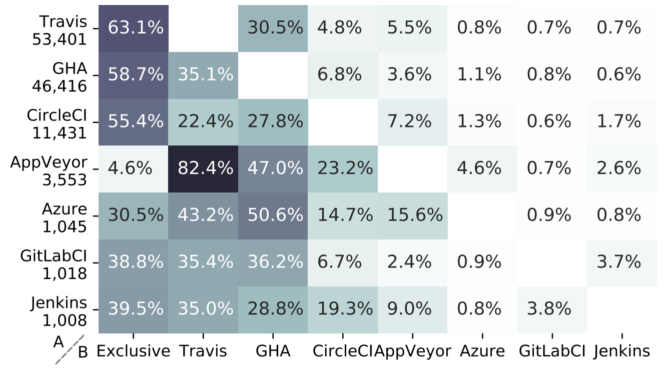

Fig. 4: Proportion of repositories having used CIs A and B relative to all repositories having used A.

We also report in the first column of Fig. 4 the proportion of repositories in which a CI has been exclusively used. We observe that a majority of the repositories have exclusively used Travis (63.1%), GHA (58.7%) or CircleCI (55.4%). The opposite can be observed for AppVeyor, Azure, GitLab CI and Jenkins. Out of the 1,045 repositories having used Azure, more than half of them (50.6%) also have used GHA. This is even more pronounced for AppVeyor: only 4.6% of the 3,553 repositories have used it exclusively, while 82.4% of the repositories have also used Travis, 47.0% have also used GHA, and 23.2% have also used CircleCI.

The majority of repositories having used multiple CIs involve Travis, GHA and CircleCI. Those same CIs are also the ones that are most frequently used *exclusively* in the considered repositories. AppVeyor is very rarely present exclusively in a repository.

So far, we considered pairs of CIs, disregarding whether the CIs were used *simultaneously*. Let us therefore consider the ***co-usage*** of CI pairs in a given repository, defined as those situations where the repository uses both CIs for a common period of at least 30 days. If a repository would simultaneously use 3 different CIs A, B and C, this will be counted as 3 co-usages, namely for each pair (A, B), (B, C) and (A, C).

We found 14,335 co-usages in 11,049 distinct repositories out of 22,877 repositories having used multiple CIs during their lifetime. Travis, AppVeyor, GHA and CircleCI are involved in most co-usages, together covering 86.7% of all co-usages and 92.1% of all repositories with co-usage.

Fig. 5 shows the proportion of co-usages we found for each pair of CIs A and B, relative to all repositories involving A. We also report in the first column the proportion of cases with no co-usage. For all CIs except AppVeyor, more than half of the repositories using them do not use another CI at the same time. Travis, GHA and CircleCI are considerably less frequently used in combination with another CI than the others CIs. At the other extreme, only 6.2% of the usages of AppVeyor do not involve another CI at the same time.

Focusing on the actual cases of co-usage (i.e., all but the first column), AppVeyor is mostly co-used with Travis (80.5%), and to a lesser extent with CircleCI (19.3%) and GHA (17.9%). Travis is the most frequent complementary CI for GHA, AppVeyor, GitLab CI and Jenkins, and GHA is most frequently complemented by Travis, CircleCI and Azure.

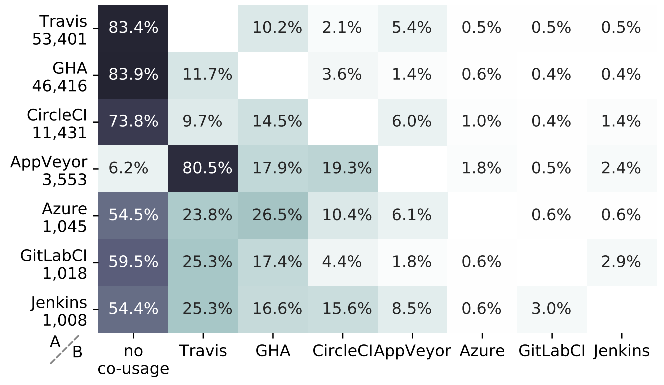

Fig. 5: Proportion of co-usages of CIs A and B relative to all repositories using A.

With the notable exception of AppVeyor, the majority of the repositories with a CI do not use another CI simultaneously. The overwhelming majority of repositories using AppVeyor co-use Travis along with. Travis and GHA are the most frequent complementary CIs.

V. RQ_3: How frequently are CIs being replaced by an alternative?

The findings of RQ_2 revealed a considerable amount of combinations of multiple CIs within repositories. Part of these combinations were due to ***co-usages***. Another likely scenario is that a repository has used multiple CIs during its lifetime, but not necessarily simultaneously. This could be the case, for example, when a repository maintainer is unsatisfied with its current CI and decides to replace it with another CI that offers solutions that better meet the needs of the team or the project.

We consider that a repository ***migrated*** from some CI A to another CI B if the repository stopped using A and started using B at "around the same time", i.e., within a time window of 30 days either *before* or *after* B started to be used by the repository. The rationale behind this time window is to accommodate for a possible transition period during which either both CIs were being used together (i.e., time needed to remove A after the introduction of B) or where none of them was being used (i.e., time needed to introduce B after the removal of A).

We detected 14,219 such cases of migration in 13,083 different repositories. Table III reports on the number of repositories involved in a migration away from or towards a given CI. It also reports on their proportion relative to the total number of repositories using a given CI. We observe that most migrations (11K+) are due to repositories migrating away from Travis and, to a lesser extent, from CircleCI. On the other hand, we see that most migrations (12K+) target GHA and, to a lesser extent, again CircleCI. For most of the

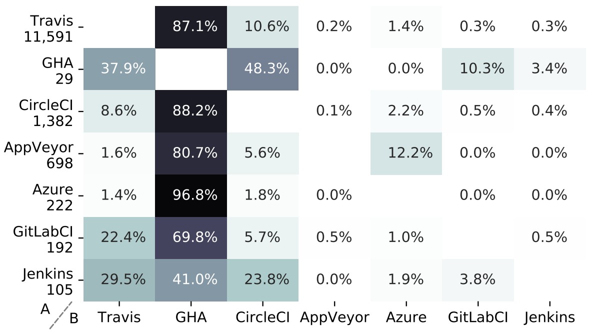

Fig. 6: Proportion of migrations from CI A to CI B, relative to the total number of migrations away from A.

CIs, we observe there are more migrations away from them than migrations towards them, with the notable exceptions of GHA that attracted far more repositories, and of CircleCI and Azure, both having roughly the same number of repositories that migrated from and to them. Looking at the proportions of repositories relative to the total number of repositories using a given CI, we see that around one out of five repositories using Travis, AppVeyor, Azure or GitLab CI migrated to another CI. On the other hand, only 29 out of the 46,416 repositories using GHA migrated away from it. The last column reveals that migrations explain more than one out of four repositories using GHA or Azure. In contrast, less than 1% of the repositories using Travis or AppVeyor are due to a migration.

Most migrations are from Travis and towards GHA. The number of repositories migrating away from CircleCI and Azure is balanced with the number of repositories migrating towards them. Migrations towards GHA and Azure account for one fourth of the repositories using them.

Going into the details of these migrations between CIs, Fig. 6 shows the proportion of migrations from CI A to CI B relative to the total number of migrations away from A. This allows to see the distributions of the target CIs for migrations originating from A. We observe that GHA proportionally represents the vast majority of the targets of a migration, regardless of the considered source CI. GHA attracted up to 96.8% of all migrations away from Azure. The only notable exception is Jenkins: even if GHA still accounts for 41.0% of the migrations from Jenkins, more than half of the migrations are towards Travis and CircleCI. Travis is also the second most frequent target for migrations originating from CircleCI and GitLab CI.

TABLE III: Number of repositories that migrated away from and towards a CI, and their relative proportion.

CI	repo.	migrated away from #	%	migrated towards #	%
Travis	53,401	11,591	21.7%	218	0.4%
GHA	46,416	29	0.06%	12,269	26.4%
CircleCI	11,431	1,382	12.1%	1,323	11.6%
AppVeyor	3,553	698	19.6%	30	0.8%
Azure	1,045	222	21.2%	282	27.0%
GitLab CI	1,018	192	18.7%	53	5.2%
Jenkins	1,008	105	10.4%	44	4.4%

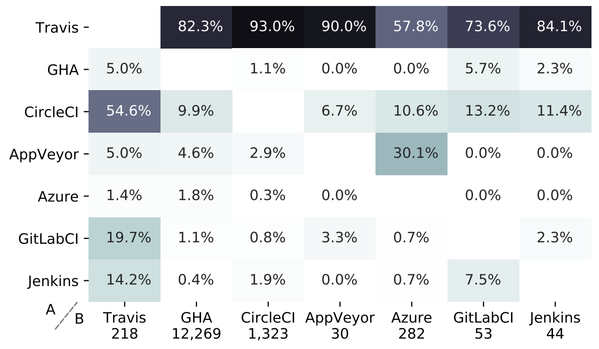

Fig. 7: Proportion of migrations from CI A to CI B, relative to the total number of migrations towards B.

GHA attracted most of the migrations, regardless of the source CI. Travis is the second most frequent target for migrations from CircleCI, GitLab CI and Jenkins.

Fig. 6 provided insights about the *target* of CI migrations. To provide insights about the *source* of CI migrations, Fig. 7 reports on the proportion of migrations from CI A to CI B, this time relative to the total number of migrations towards B. We observe that Travis provided the overwhelming majority of the repositories that migrated to another CI, regardless of the target CI. It represents up to 93.0% of all migrations towards CircleCI, accounting for 1,230 migrations. On the other hand, even if CircleCI accounts for more than half of the migrations towards Travis, this corresponds to 119 migrations only. CircleCI is also the secondmost frequent source of migrations for all CIs, except for Azure whose secondmost frequent source of migrations is AppVeyor. Nearly one third of all migrations towards Azure originate from AppVeyor.

Travis provided the vast majority of the repositories that migrated, regardless of the target CI. CircleCI is the second most frequent source of migrations towards Travis, GHA, AppVeyor, GitLab CI and Jenkins.

VI. RQ_4: How has the CI landscape changed since GHA was introduced?

RQ_1 revealed that, since the public release of GHA in November 2019, its market share has been increasing very fast, becoming the dominating CI in less than 18 months time! GHA hence seems to have played a major role in how the CI landscape had changed, with many repositories co-using GHA with some other CI (RQ_2), or even migrating towards GHA (RQ_3). The increase in popularity of GHA seems to have had a diminishing effect on the share of repositories using Travis and other CIs. RQ_4 therefore aims to scrutinise to which extent GHA has altered the CI landscape.

To study the effect of the introduction of GHA on the usage of other CIs, we use the statistical technique of (linear) Regression Discontinuity Design (RDD) [8], [9]. This technique allows to model the effect of a particular important event by comparing the situation during a given time window before and after the event. In our case, we intend to use RDD to model the effect of the introduction of GHA on CI usage of repositories before and after this event. We consider August 8, 2019 as the event date since it corresponds to the day at which GHA announced the availability of a CI/CD service. RDD assumes that one would be able to observe a discontinuity in the data if the event affects the outcome (here, CI usage). Such a discontinuity would reveal itself as a perceptible difference in the intercept and/or slope of the (linear) regression model before and after the event. The RDD model is formulated as:

$$y_i = \alpha + \beta \times \text{time}_i + \gamma \times \text{event}_i + \sigma \times \text{time\_after}_i + \epsilon_i$$

where i corresponds to an observation related to a given CI before and after the event. In order to incorporate the passage of time into the model, three parameters are used: *time*$_i$, *event*$_i$ and *time_after*$_i$. The *time* parameter is the number of months that have passed from the beginning of the observation window. We use two observation windows of 12 months each, respectively until 1 month before and starting from 1 month after the event. We purposefully ignore what happened between 30 days before and 30 days after the event to account for possible instabilities close to the event date. The binary *event* parameter specifies whether an observation is measured before (event $= 0$) or after the event (event $= 1$). The *time_after* parameter indicates elapsed time (expressed in number of months in our case) since the event and it is set to 0 before the event. ϵ_i is the residual error. The two resulting linear regression lines have a slope of β before the event, and $\beta + \gamma$ after it. The difference between the two regression values of y_i indicates the size of the effect of the event. The accuracy of the RDD model is estimated using R^2, a common method for assessing the goodness of fit of a regression model. We implemented the RDD model based on the ordinary least square method using the statsmodels library in Python.

Given our aim to determine the effect of the public release of GHA on the usage of other CIs, we computed for each CI the monthly variation of CI usage, measured as the number of repositories adopting the CI minus the number of repositories discontinuing the CI. We fit 7 different RDD models: one for assessing the global effect of the event on the CI landscape, and one for the individual effect on each of the 6 most popular CIs (excluding GHA itself). The results are reported in Table IV. We provide the R^2 goodness of fit of each model, the coefficients of each model parameter, and the statistical significance of the coefficients in terms of their p-value. We consider three levels of significance, reflecting the strength of the effect induced by the event: $p < 0.001$ for strongly significant (***), $p < 0.01$ for highly significant (**), and $p < 0.05$ for moderately significant (*).

Only 4 out of the 7 computed RDD models provide a sufficiently high goodness of fit: CI landscape, Travis, Azure and CircleCI (in decreasing order of goodness of fit, with R^2 ranging from 0.94 to 0.57). We will therefore only discuss these 4 models in more detail. Only for Travis ($p < 0.001$), Azure ($p < 0.001$) and CircleCI ($p < 0.01$) we observe a

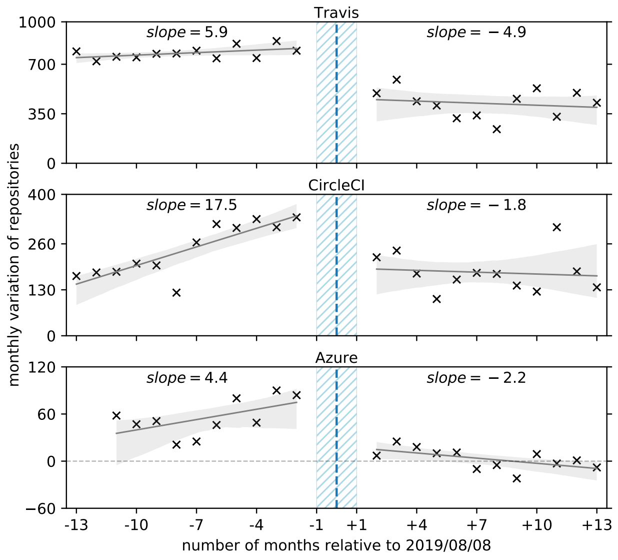

Fig. 8: RDD analysis on the monthly variation of repositories making use of each CI (= adoptions - discontinuations) before and after introduction of GHA.

statistically significant effect of the event. The coefficients of *time* and *time_after* are significant for CircleCI and Azure. These coefficients indicate that the introduction of GHA had a significant impact on Travis, Azure and CircleCI usage.

To observe the effects of the introduction of GHA, Fig. 8 provides a visualisation of the RDD models for Travis, CircleCI and Azure. The figure confirms the observations of the statistical analysis, revealing a decline in the monthly variation of CI usage for each of them. We also observe a change in the direction of the slope (from positive to negative), suggesting that after the event the growth rate of CI usage decreases. Still, for Travis and CircleCI, the number of adoptions remains higher than the number of discontinuations. For Azure, however, the monthly CI usage variation start to become negative after the event, implying that more repositories discontinue Azure compared to those repositories adopting it.

The introduction of GHA aligns with a decreasing growth rate of CI usage for Travis, CircleCI and Azure.

VII. Discussion

The rise of GitHub Actions

Perhaps the most surprising result of our empirical analysis is that GHA has become so popular so quickly. In less than 18 months time, GHA has changed the CI landscape completely, becoming the dominating CI at the expense of Travis that had been dominating the landscape for more than nine years. A deeper quantitative and qualitative analysis might uncover the reason for the popularity of GHA, but we can already emit a number of reasons and hypotheses that deserve further study:

- GHA is fully integrated with GitHub, obviating the need to resort to external CI services;

TABLE IV: RDD analysis of the monthly variation of CI usage, before and after official introduction of GHA.

CI	**event** (error)	**time** (error)	**time_after** (error)	**constant** (error)	R^2
CIs landscape	146.0 (172.4)	25.8 (17.6)	**132.1***** (24.8)	**943.8***** (129.3)	0.94
Travis	**-357.1***** (64.5)	5.9 (6.6)	-10.8 (9.3)	**741.0***** (48.4)	0.86
CircleCI	**-148.8**** (41.4)	**17.5***** (4.2)	**-19.3**** (6.0)	**128.5***** (31.1)	0.57
AppVeyor	-16.5 (20.1)	-1.4 (2.1)	-0.1 (2.9)	24.2 (15.1)	0.38
Azure	**-57.8***** (13.6)	**4.4*** (1.8)	**-6.6**** (2.2)	22.2 (14.1)	0.79
GitLab CI	-0.7 (4.6)	-0.4 (0.5)	-0.4 (0.7)	**23.1***** (3.5)	0.46
Jenkins	17.6 (10.8)	-2.2 (1.1)	2.4 (1.6)	**22.6*** (8.1)	0.22

*** p <0.001, ** p <0.01, * p <0.05

- GHA is trendy, very easy to set up and use, and directly accessible to any GitHub repository through the Actions tab;
- GitHub comes with a large marketplace, allowing repository maintainers to select the Actions they need, and allowing Action developers to easily adapt existing Actions, or create new ones based on predefined templates;
- GHA provides a generic mechanism for automating anything in the GitHub development workflow, making it more general than "just" a CI/CD service. GHA is rather feature-complete, allowing to automate not only commit-related activities but also activities related to comments, issues, pull requests and many more.

A parallel line of future work, similar in spirit to the work of Zampetti et al. [10] on Travis pipelines, would consist of a fine-grained analysis of how GitHub Actions and their usage evolve over time. In the same line, we could study the effort needed to adopt or migrate to a given CI, how easy it is to modify a given CI usage, and so on. Such analysis would require analysing the commits touching the CI-related files, the number of involved contributors, etc.

The fall of Travis

During most of the observation period of our analysis, Travis was dominating the CI landscape, used by more than four out of five repositories up to January 2019. Since November 2020, nearly two years after the introduction of GHA, however, we observed an unexpected and sudden decrease in the number of usages of Travis (see Fig. 2). This important change trend coincides with the decision of Travis, on November 2nd 2020, to alter its free plan for public repositories[2] coinciding with a sudden increase in discontinuations of Travis combined with a progressive decrease of adoptions of Travis (cf. Fig. 3).

According to the announcement, the change in pricing structure has been decided in order to get shorter build times and greater flexibility. However, at the same time, Travis decided to impose some restrictions on the free plan. First, they dropped support for MacOS builds. Second, they limited the overall number of concurrent builds across *all* open-source projects to around 560 jobs, with the consequence of much longer build delays, especially during the day. Lastly, they imposed a limit of 1,000 minutes of (cumulated) build execution time for all Linux and Windows builds, even for

[2]https://blog.travis-ci.com/2020-11-02-travis-ci-new-billing

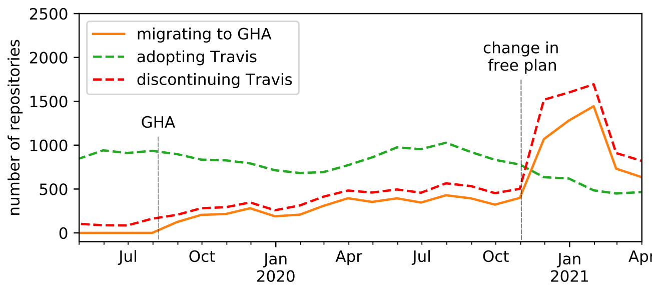

Fig. 9: Monthly number of repositories adopting Travis, discontinuing Travis and migrating from Travis to GHA.

open-source projects. Even if the quota can be refurbished on demand, these changes to the free plan have sparked numerous complaints from maintainers of open-source projects.[3]

We investigated whether Travis' decision resulted in a large number of repositories migrating away from Travis to a competing CI. Fig. 9 shows the monthly number of repositories adopting and discontinuing Travis (as seen in Fig. 3), as well as the number of repositories that migrated from Travis to GHA. This figure shows that Travis' change in free plan drove many repositories to discontinue Travis and that most of them migrated to GHA, much more than when GHA was introduced. For instance, we found that 3,793 repositories migrated away from Travis to GHA within the 3 months following Travis' change in free plan, accounting for more than one third (37.5%) of all migrations from Travis to GHA since its introduction 15 months before. For comparison, only 1,115 repositories migrated during the 3 months preceding the change (11% of all migrations).

One should nevertheless note that, despite the many migrations and discontinuations from Travis, it continues to be one of the most popular CI services, remaining the secondmost used CI for GitHub repositories of npm packages.

Co-usage of multiple CIs

In RQ_2 we observed that around 12% of all repositories use multiple CIs simultaneously. This is surprising since one might intuitively expect all CIs to provide similar services.In order to understand the reasons behind the use of multiple CIs, we contacted the maintainers of some of the repositories in our dataset that were found to co-use CIs. Based on the answers

[3]https://pastebin.com/2Y7LY0EN compiles some supporting references.

they provided, we identified several reasons why a repository might rely on multiple CIs at the same time.

Firstly, most CIs have limitations in the number of jobs that can be run in parallel, which can significantly increase the time needed for builds. Using multiple CIs allows to benefit from a kind of load balancing at the level of the builds. One maintainer reported: "*runtime balancing at the DNS level is not enough for me. I need that also at build time.*"
Job execution by multiple CIs was mentioned by a maintainer as a more fault tolerant approach: "*my git repo is built and tested against at least 3 remotes. I occasionally see any one of them down once in a few months or so. But more importantly, my own pipeline sometimes has specific parts failing on one or more of those providers. One ultimately always passes so that my nightly build is always safe and done*".
Another reason to use multiple CIs relates to the set of supported operating systems. Although most CIs support today's main operating systems, this has not always been the case. For example, AppVeyor was the only major CI offering Windows builds up to 2018. At the same time, AppVeyor was known to have strong limitations in its free plan, such as no concurrent jobs or long build queues.[4] Therefore, repositories wishing to benefit from Windows support *and* fast builds for Linux had no choice but to combine AppVeyor with another CI. This may explain why we found, for an overwhelming majority of AppVeyor usages, repositories co-using Travis at the same time. Interestingly, the decline of AppVeyor observed in RQ_1 coincides with the availability of Windows builds in competing CIs. Indeed, Travis started to support Windows builds in October 2018,[5] shortly after Azure. For comparison, support for Windows was added in August 2019 for CircleCI and in January 2020 for GitLab, and was already available in GHA since its release in November 2019.

As a follow-up research, we aim to complement our quantitative analysis by more qualitative insights (e.g., by conducting developer surveys) in order to better understand the reasons that drove developers to use multiple CIs simultaneously and the implications of doing so, to identify the reasons and the benefits of migrating from a given CI to another one. With such additional insight, we plan to create recommendation models to suggest repository maintainers when to complement a given CI and by which one, as well as when and how to migrate to another CI and which one.

VIII. Threats to Validity

Using the framework of Wohlin et al. [11], we discuss the threats that may affect the validity of our findings.

Construct validity: As any empirical study focusing on GitHub project histories, the validity of our findings is subject to the intrinsic limitations of mining git [6] and GitHub [4]. Another important threat concerns the accuracy of our dataset. We detected CI usage in GitHub repositories on the basis of the presence of specific CI-related files. This is an underestimation, since it precluded us from detecting CIs that are self-hosted using a locally configured service, that are configured through their web interface (e.g., AppVeyor, Jenkins, Amplify or Buddy), or that do not impose a specific file path for their configuration files (e.g., Buildkite). This well-known problem has been reported by others as well [12] and it explains why the use of Jenkins is underrepresented in our dataset. In fact, we have only been able to detect usages of Jenkins since the introduction of its pipeline-as-code feature in April 2016.

To some extent, our dataset is also subject to overestimation, since the presence of CI-related files does not necessarily imply that the CI is actually being used and triggers any build. It is also likely that we overestimated the use of GHA as a CI. We assumed that any use of GHA relates to continuous integration, while it can serve different purposes such as welcoming users, closing issues, etc. However, this is unlikely to affect our findings, since the vast majority of the available Actions on the GitHub marketplace relate to continuous integration, and since our manual inspection of several repositories did not reveal any case where GHA is used exclusively for tasks unrelated to continuous integration.

Conclusion validity: Since our conclusions are mostly based on quantitative observations, they are unlikely to be affected by such threats.

Internal validity: We set a threshold of 30 days to detect CI usages in repositories to reduce the possible noise introduced by short-lived tests or attempts to integrate a CI. We used the same threshold to distinguish between co-usages and migrations, notably to accommodate for a transition period during migrations. We manually checked how variations in this threshold would affect the reported findings. Despite some expected small variations on the observations we made, the findings remain the same.

External validity: Our findings do not necessarily generalise beyond the considered dataset of GitHub repositories for active npm packages. The results may differ for software repositories belonging to other package distributions; for arbitrary GitHub projects; for software repositories using other online coding platforms (e.g., GitLab); and for other versioning systems. Replications of our analysis beyond the current dataset would be needed to check whether the observed findings generalise in a wider context.

IX. Related Work

To the best of our knowledge, Kinsman et al. [13] are the only researchers having empirically studied GHA. Their analysis focused on the adoption of GHA by GitHub repositories. Less than 1% of the considered active repositories were found to adopt GHA, and only 926 projects in their dataset had adopted it for at least 6 months. Our own analysis, based on more recent data, shows a quite different story, with a dramatic increase in popularity of GHA in 2020 and 2021. The research questions targeted by our analysis are complementary, since they focused on a coarse-grained analysis to compare GHA against other CIs, whereas [13] carried out a fine-grained

[4]https://pastebin.com/RAH0wFQ5 compiles some supporting references.
[5]https://blog.travis-ci.com/2018-10-11-windows-early-release

analysis to categorise the types of actions being used and discussed, as well as their impact on pull requests.

Given its popularity, the use of Travis in GitHub projects has been a prolific empirical research topic [10], [12], [14]–[18]. For example, Vasilescu et al. [12] empirically studied Travis usage in 918 out of 1,884 GitHub projects (corresponding to a usage ratio of 48.7%), extracted in November 2014. They observed that projects using Travis have higher productivity in terms of more pull requests processed (accepted, merged and rejected) without an observable decrease in quality (in terms of reported bugs). As another example, Cassee et al. [15] studied the impact of Travis on code reviewing practices in 685 GitHub projects. They observed that in projects using Travis, pull requests are having less discussions, suggesting that the same amount of work can be carried out with less need for interdeveloper communication. Beller et al. [16] focused on the relation between testing and Travis usage, by analysing failures in 2,640,825 Java and Ruby builds. Zampetti et al. [10] conducted a fine-grained analysis of how specific Travis pipelines evolve and get restructured over time. In line with our quantitative observations of the fall of Travis in GitHub projects, Widder et al. [19] identified, through a mixed methods study, eight main reasons why projects decided to abandon the use of Travis: long build times, unsupported technologies, CI consistency across projects, lack of tests, infrequent changes, poor user experience, closed source concerns, and troubleshooting build failures. Given that the study was conducted before the official support of CI through GHA, the migration towards GHA was not identified as a main reason.

In their seminal blog [20], Fowler and Foemmel introduced 10 core CI practices to increase the speed of software development. Since then, dozens of open source or commercial CI tools and services have been introduced and used. Elaszhary et al. [21] studied the benefits and challenges of these 10 core CI practices in three software-producing companies using them. These practices were broadly implemented with quite some variation, calling for further studies to understand these differences and their impact on software quality and process improvement. Specifically focusing on npm package repositories on GitHub, Lamba et al. [22] studied the spread of CI and quality assurance tools over time. They found social factors to contribute significantly to the probability of tool adoption, with characteristic differences between early and late adopters. They also observed that repositories tend to stick to a given CI tool once it has been adopted.

Hilton et al. [7] carried out a mixed-methods study of CI usage in GitHub projects. They surveyed 442 developers and analysed 35,555 GitHub projects and 1,529,291 builds to gain insights in the costs and benefits associated with CI. They observed that CIs are widely used, with Travis (90.1%) and CircleCI (19.1%) being the most popular by far, and a median adoption rate of one year. They also hypothesised that the CI adoption rate (40.27%) would increase even further. Our more recent analysis did reveal a higher CI adoption rate (53.61% in May 2021) but we did not observe an increase in adoption rate over time. We could confirm that Travis and CircleCI were among the top 3 most popular CIs, but GHA emerged as a newcomer in the CI landscape that has overtaken all other CIs in popularity.

X. Conclusion

In this article we reported on an exploratory census of the evolving landscape of CI services used in GitHub repositories for npm. Specific changes in policies and features offered by existing CIs, as well as the introduction of new CIs, has lead to important shifts in CI popularity and usage. To show this, we conducted a longitudinal quantitative analysis of CI co-usage and migration in 91,810 GitHub repositories corresponding to the nine-year development history of npm packages. We focused on the seven most popular CI services covering 99% of all observed usages: Travis, GitHub Actions (GHA), CircleCI, AppVeyor, Azure, GitLab CI and Jenkins.

Noticing that 1 out of 4 repositories has used more than one CI during its lifetime, we analysed the simultaneous usage of multiple CIs. This revealed a frequent co-usage of AppVeyor with other CI services in order to enable builds for the Windows platform. Yet, several limitations of AppVeyor and improvements in competing CIs have lead many repositories to stop using AppVeyor.

Travis and GHA were the predominant CIs, having been used in over 90% of the considered repositories. We found objective evidence for the rise of GHA, becoming the most dominant CI in less than 18 months, taking over the role of Travis that dominated the scene for more than nine years. Using regression discontinuity design we determined that introduction of GHA aligned with a decreasing growth rate in the usage of Travis, CircleCI and Azure. We also found evidence for the fall of Travis due to a combination of multiple factors. The introduction of GHA as a new CI caused many repositories to migrate from Travis to GHA. The decision to change the free plan policy of Travis further increased migrations from Travis to one of its competitors.

Our results are based on empirical observations, providing insights in how the Github CI landscape has been experiencing important changes. If the empirical findings should generalise beyond npm, they provide rich information to OSS practitioners on the most appropriate CI to adopt as well on which CIs to stop using. Maintainers of CI solutions can benefit from it as well, to counter negative effects of implementation choices and policy changes, as well as to cope with the impact of competing services, in order to survive in the rapidly evolving CI landscape. Our research also opens the door for future qualitative analyses in order to answer important research questions such as: What are the reasons why projects migrate between CIs? How easy is it to perform such a migration? What are the consequences of those migrations?

Acknowledgment

This research is supported by the Fonds de la Recherche Scientifique – FNRS under Grants number O.0157.18F- RG43 (Excellence of Science project SECO-ASSIST) and T.0017.18.

References

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *International Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2012, pp. 1277–1286.

[2] C. Chandrasekara and P. Herath, *Hands-on GitHub Actions: Implement CI/CD with GitHub Action Workflows for Your Applications*. Apress, 2021.

[3] GitHub, "The 2020 state of the octoverse - community report," 2020. [Online]. Available: octoverse.github.com

[4] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 92–101.

[5] J. Katz, "Libraries.io open source repository and dependency metadata (version 1.4.0) [data set]," http://doi.org/10.5281/zenodo.2536573, 2018.

[6] C. Bird, P. C. Rigby, E. T. Barr, D. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *Working Conference on Mining Software Repositories (MSR)*. IEEE Computer Society, May 2009. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-promises-and-perils-of-mining-git/

[7] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 426–437.

[8] D. L. Thistlethwaite and D. T. Campbell, "Regression-discontinuity analysis: An alternative to the ex post facto experiment." *Journal of Educational Psychology*, vol. 51, no. 6, pp. 309–317, 1960.

[9] T. D. Cook and D. T. Campbell, *Quasi-experimentation : design & analysis issues for field settings*. Boston: Houghton Mifflin, 1979.

[10] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2021.

[11] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer, 2012.

[12] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in GitHub," in *Joint Meeting on Foundations of Software Engineering (FSE)*, 2015, pp. 805–816.

[13] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use GitHub actions to automate their workflows?" in *International Conference on Mining Software Repositories (MSR)*, 2021.

[14] B. Vasilescu, S. van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. van den Brand, "Continuous integration in a social-coding world: Empirical evidence from GitHub," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 401–405.

[15] N. Cassee, B. Vasilescu, and A. Serebrenik, "The silent helper: The impact of continuous integration on code reviews," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 423–434.

[16] M. Beller, G. Gousios, and A. Zaidman, "Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub," in *International Conference on Mining Software Repositories (MSR)*, 2017, pp. 356–367.

[17] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on GitHub," in *Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 367–371.

[18] 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 *International Conference on Automated Software Engineering (ASE)*, 2017, pp. 60–71.

[19] D. G. Widder, M. Hilton, C. Kästner, and B. Vasilescu, "A conceptual replication of continuous integration pain points in the context of Travis CI," in *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2019, pp. 647–658.

[20] M. Fowler and M. Foemmel, "Continuous Integration," https://martinfowler.com/articles/originalContinuousIntegration.html, September 2000, [Online; accessed 3 January 2022].

[21] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M.-A. Storey, "Uncovering the benefits and challenges of continuous integration practices," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021.

[22] H. Lamba, A. Trockman, D. Armanios, C. Kästner, H. Miller, and B. Vasilescu, "Heard it through the gitvine: An empirical study of tool diffusion across the npm ecosystem," in *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2020, pp. 505–517.