Production Systems and Information Engineering
Volume 10 (3), pp. 19–25
doi: 10.32968/psaie.2022.3.2

AN EXPLORATIVE ANALYSIS OF MANAGED CI/CD USAGE AMONG OPEN-SOURCE C/C++ PROJECTS

Áron Kiss
University of Miskolc, Hungary
Institute of Information Technology
kiss.aron@uni-miskolc.hu

Abstract. CI/CD is a common practice in software projects today, because it provides a higher level of reliability and safety, especially in the case of dynamically typed script languages. Several studies have examined questions about the effects of using CI/CD pipelines in general. This paper presents, how projects are written primarily in one of the mature C and C++ programming languages adapt to the emerging CI/CD trend. What proportion of these projects are using a CI/CD pipeline? Which managed CI/CD services are typically used in these projects? How early are the CI/CD approach adopted? Is project popularity related to CI/CD adoption rate? Data about 6,000 open-source C/C++ repositories were collected and analysed in this study to answer the aforementioned questions.

Keywords: continuous integration, agile software development, software quality assurance

1. Introduction

Continuous Integration, Delivery and Deployment are modern approaches, that make it possible to automate linting, building, packaging, testing, releasing, and deployment of software products. This provides more reliable software and reduces the need for human labor during software release or deployment to a production system.

Nowadays, many software support the needs of software developers in connection with their CI/CD process. There are on-premise and managed solutions on the market. Most of the modern CI/CD solutions can be integrated with code hosting platforms, like GitHub.

Nowadays CI/CD methods are actively researched area of software engineering. [1] concluded that adoption of CI improves the productivity of the development team and makes it possible to integrate more outside contribution without degrading code quality. [2] was combined their data set of open-source software projects with an online survey to find out why developers chose to use or not to use CI, and what are the benefits and costs of introducing a CI/CD pipeline to a project. [3] and [4] are examined automated build runs in Travis CI to answer questions about the usage characteristics of managed CI/CD solutions.

Studies are reported that CI/CD pipeline is more often used in projects that are written in modern, dynamically-typed languages, such as Python, JavaScript and Ruby [2, 4].

However, I did not find studies that specifically examine the characteristics of CI/CD use among software developed with more mature, but still widely used programming languages, especially C and C++. The aim of this study is to explore, how C/C++ developers are adopting CI/CD in their projects, and what characteristics can be observed in their CI/CD usage. By collecting and analysing a set of open-source C and C++ projects, I will be able answer several research questions:

RQ1. How common is the usage of managed CI/CD services among C and C++ projects?
RQ2. Which managed CI/CD solutions are used for open-source C/C++ projects in GitHub?
RQ3. How early do C/C++ developers adopt CI/CD in their projects?
RQ4. How does the popularity of projects influence the CI/CD usage rate?

2. Background

CI/CD – which is an acronym of Continuous Integration and Continuous Delivery or Continuous Deployment – is a collection of principles and technologies that make it possible to develop and release software products faster and more efficiently, than following classical software engineering approaches like the waterfall model. CI/CD's aim is to help software development and operation by introducing automated processes, which are triggered by development units (e.g. commits in the version control system [VCS]). These automated processes can carry out multiple checks, formatting and inspection (e.g. linting code based on pre-defined rules, fetching dependencies, building binaries, unit testing, integration testing, acceptance testing, etc.) on the codebase, can build manually releasable units of the software and can deploy a release of the software to production environments completely automatically.

Continuous Integration (CI) is the process during which every single commit pushed to the central repository is evaluated with a test suite and merged into the main development branch only if all the test cases ran successfully [2].

Continuous Delivery (CDE) is the process of building a manually installable unit from the new state of the codebase. CDE often includes steps like compiling and packaging the code, running unit-, integration- and automated acceptance tests. This ensures that the software can be released reliably at any time [5].

Continuous Deployment (CD) is the process where the changes of the codebase — even the smallest ones – are delivered frequently through completely automated deployments to production or customer environments [6]. We can consider CD as a more complete form of CDE, as during this process not only the deployable software unit is produced, but it is also deployed automatically.

CI, CDE and CD processes are often depending on each other's output, so we can collectively refer to this automation as a *CI/CD pipeline* [6].

By introducing a CI/CD pipeline to a project, errors can be traced back to smaller code changes, shortly after the changes are published to the VCS. As a result, the cost of fixing errors and the chance to introduce new ones can be reduced.

This practice of merging fewer changes at a time allows developers to resolve merge conflicts earlier and address regression defects more efficiently and quickly [7]. However, setting up the CI/CD environment and later operating and maintaining it is a time-consuming task that requires human work. This is partially eliminated by today's cloud-based solutions. Secondly, the method itself can only add unnecessary complexity to smaller projects. [8] found that very small projects (up to 1,000 lines of code) are the ones that take the longest time to fix broken builds. Moreover, CI/CD's advantages are not necessarily noticeable if one want to apply it to a legacy codebase that cannot be covered well by automatic tests.

2.1. Basic example of CI/CD workflow

Figure 1 shows a basic CI/CD workflow. In step 1, a developer pushes changes to the central repository.

In step 2, the CI/CD service has to be notified about the code change. It can be achieved in two ways: a) the repository service notifies the CI/CD server about the change (e.g. GitHub Webhooks); b) the CI/CD server polls the state of the central repository if any changes are made [9].

In step 3, the CI/CD server fetches code from the central repository and run the pre-defined tasks on it. These tasks can include but not limited to

- linting the code,
- compiling the code,
- packaging the output of compilation,

- running automated unit, integration, and acceptance tests,
- determining code quality,
- measuring code coverage,
- generating documentation,
- releasing executable binaries,
- deploying the software automatically.

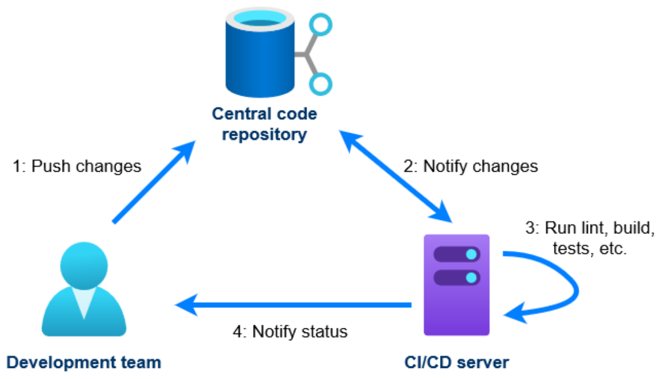

Figure 1. Basic CI/CD workflow

In step 4, the CI/CD server sends report about the process to the development team and/or the code hosting service. This report can contain information such as metadata about the testing environment, logs of the standard output, and the resulting state of the CI/CD pipeline's actual run (success, failure, timeout, etc.). In case of error, the VCS provider can automatically decline merging the changes to the main development branch.

2.2. On-premise and managed CI/CD solutions

CI/CD solutions can basically be divided into two groups based on their operating method.

On-premise CI/CD servers (e.g. Jenkins, TeamCity, Bamboo, Azure DevOps) are operated by the organization that uses the CI/CD pipeline in their development process. Setting up the testing environment, and later operating and maintaining it is the organization's responsibility, which requires human work. When CI/CD is heavily used in the development of software systems, a new role may even be involved, which is often called *DevOps Engineer*. DevOps Engineers are responsible for reducing the time between committing a change to a system and the change being placed into normal production [10]. This can be achieved by building an effective pipeline of releases.

Managed CI/CD solutions (e.g., GitHub Actions, Azure Pipelines, TravisCI, CircleCI) are hosted, scaled, secured and generally maintained by an external organization. These external organizations are only provide CI/CD capabilities to the development teams through APIs. This can offload a large amount of work from the development team. However, managed CI/CD services are often considered less secure than on-premise solutions, because the software product's code is examined and executed in an external company's infrastructure, most often in a multitenant environment, where malicious tenants can damage the CI system as a whole or can compromise builds of other tenants [11]. This might also cause compliance issues with organizational standards and further regulations in certain cases (e.g. in the development of healthcare software).

Developers of open-source projects are often choose managed CI/CD solutions because these are easy-to-use and often free services, at least to a limited extent [12, 13, 14]. Also, keeping the algorithm a secret is not a consideration in these projects.

3. Usage characteristics of managed CI/CD services

Here, I present the process of collecting and pre-processing the data on which the study is based. I also describe the observations made on the data.

3.1. Data collection and pre-processing

Data on which the study is based are collected from the GitHub API. This collected data is used to answer the specified questions about the usage of CI/CD pipelines in open-source C/C++ projects.

The repositories whose creation date falls between 2017 and 2022 and the C or C++ programming language was determined as the primary language were queried. For each year, the repositories was sorted in descending order of popularity (the number of stars of the repository served as the basis for determining popularity), and then data of the first 1,000 repositories was collected. These repositories are less likely to contain unmaintained, deprecated or personal-purpose (e.g. homework) projects. The resulting data set contains 6,000 records. Each record contains the name of the repository, the owner's GitHub username, the primary programming language, the creation date and time of the repository, the CI systems used in the project (if any), and the number of stars for the project.

The CI systems used in the projects are determined with GitHub's *Commit statuses API*. The last commit's status in the repository's default branch (most often called "master" or "main") was queried. The queried statuses are contained results of CI/CD pipeline(s), and also different outputs from various services (e.g. static code analysis tools, CLA bots, DCO checkers, vulnerability scanners, etc.) as structured JSON objects [15]. A commit can have several status objects (one or more for each service provider), these are returned by the API as an array. For each repository, the unique service provider names were stored as "CI systems used in the project".

The aim of this study is to examine the usage of full-fledged CI/CD services (where developers can run a completely customized pipeline) only, so services other than this have been manually deleted from the collected data. Firstly, the unique names of the service providers issuing commit statuses were queried from the data set. Each provider's name was then checked manually with Google Search, to determine if it was a full-fledged CI/CD provider that emitted the commit statuses. After that, providers of additional services were deleted from the data set.

3.2. Usage rate of managed CI/CD services among C and C++ projects is lower than in general (*RQ1*)

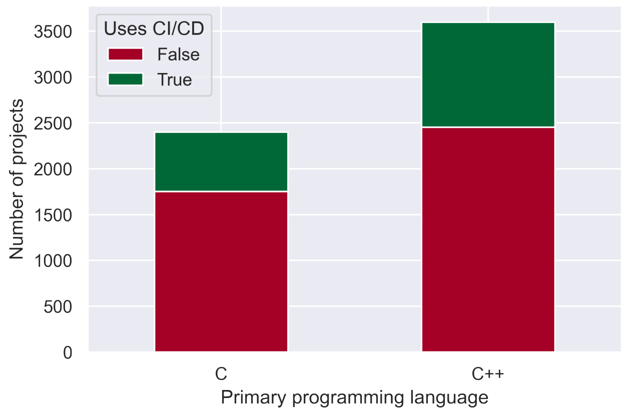

Figure 2. CI/CD usage by the primary programming language

The examined data set contains information about 6,000 GitHub repositories, of which 2,400 were C projects and 3,600 were C++ projects. It has been found, that 1,796 projects are using at least one managed CI/CD pipeline. Figure 2 shows the distribution between C and C++ projects. 27.04% of C projects and 31.86% of C++ projects are adopted CI/CD in their development process. This is significantly lower than the overall 40.27% ratio determined by [2], which refers to several programming languages, not just C and C++.

Figure 3 illustrates how many CI/CD solutions the projects are using. Most of the projects taking advantage of only one CI/CD provider, while a few projects are using 2 different CI/CD solutions at the same time. The reason for this might be that managed CI/CD services are often only free for open-source projects to a limited extent, so the CI/CD pipeline is distributed among several service providers. Another possible reason is that project owners want to test the project on several platforms/environments for greater reliability. Using CI/CD services of 3 or more providers are not typical.

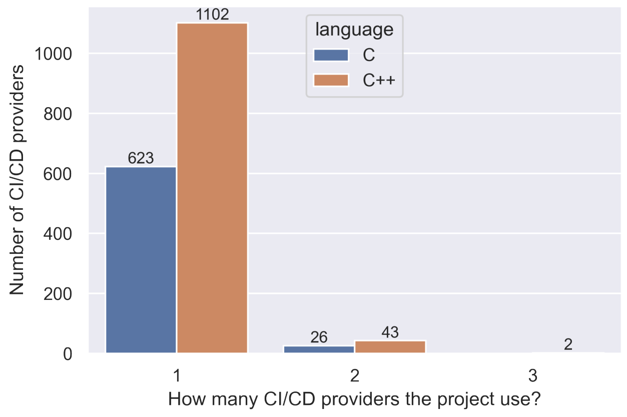

Figure 3. Number of CI/CD services used in projects

3.3. GitHub Actions is the most popular CI/CD solution used in open-source C/C++ projects in GitHub (*RQ2*)

In the collected data set, 9 of the managed CI/CD providers are presented. The exact occurences of the different providers are shown in Table 1. In the event that a project used more service providers, these were considered as one occurrence separately for each of the providers.

It is observed, that most of the CI/CD pipelines (89.6%) are run on GitHub Actions. This is related to that the data was collected from GitHub, where GitHub Actions is a closely integrated, limitedly free CI/CD alternative, that meets with the needs of most open-source projects. Usage rate of other providers are the following: Azure Pipelines 4.5%; Travis CI 3.5%; Cirrus CI 1.3%. Other CI/CD providers (Circle CI, Google Cloud Build, XCode Cloud, Multipass CI, Garnix CI) are only used sporadically. Figure 4 illustrates the distribution of occurences of CI/CD providers which are present in at least 1%. In plot a) all

Table 1. Occurences of managed CI/CD providers.

CI/CD provider	Occurences
GitHub Actions	1,672
Azure Pipelines	84
Travis CI	66
Cirrus CI	24
Circle CI	15
Google Cloud Build	2
XCode Cloud	1
Multipass CI	1
Garnix CI	1
Total:	1,866

the providers has been depicted. Plot b) shows all the providers but GitHub Actions, to exclude its distorting effect.

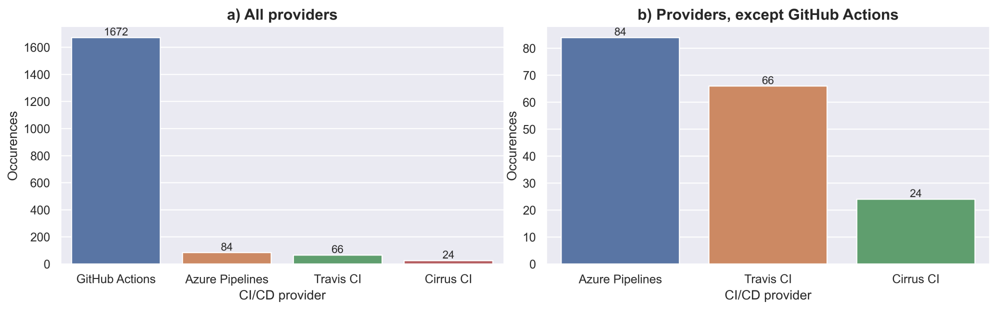

Figure 4. Managed CI/CD usage by providers

3.4. Age of project is related to the rate of CI/CD usage (*RQ3*)

The data set contains information about the 1,000 most popular repositories from each year, that are created between 2017 and 2022. Figure 5 shows the relation between the repositories' creation year and the number of repositories that are using at least one CI/CD service. Usage ratios are the following for the different years: 2017: 32.2%, 2018: 30.8%, 2019: 34.9%, 2020: 34.3%, 2021: 30.1%, 2022: 17.3%.

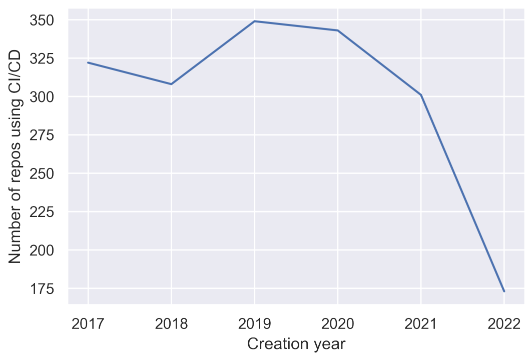

Figure 5. CI/CD usage by project age

It can be concluded from the data that less than a year old repositories are less likely to use a CI/CD pipeline. In terms of 1 to 5 years old projects, CI/CD usage rate is very similar.

3.5. Popularity of project influences the CI/CD usage rate (*RQ4*)

For this explorative analysis, the most popular repositories, created in different years are collected. Number of stars are between 43 and 85,012. Average number of the projects' stars is 934, while the median is 433 stars. 90% of the examined projects are having 1,772 or less stars, repositories with more stars than this are appear too sparsely in the data set to draw conclusions about them, so I used the bottom 90% of the projects sorted by the number of stars in descending order to answer *RQ4*. The results are illustrated in Figure 6.

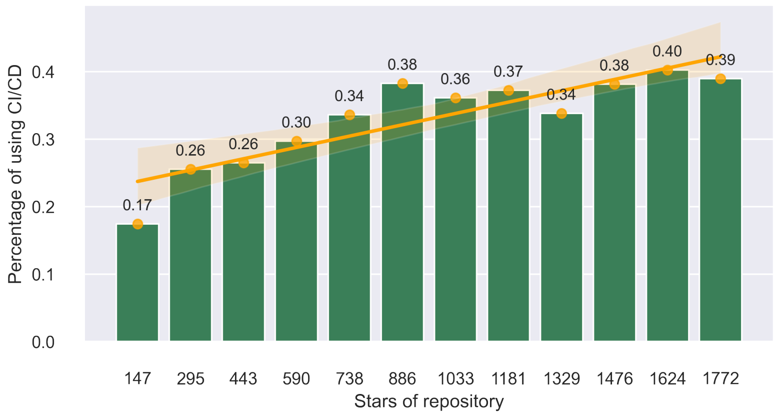

Figure 6. CI/CD usage by project popularity

Projects are sorted by popularity (number of stars) in ascending order, then divided into 12 even groups. It is observed, that the more popular a project is, the more likely it is to use at least one CI/CD pipeline. While 17% of the projects with 43–147 stars are using CI/CD, this ratio is approx. 40% for projects with 1,625–1,772 stars.

4. Conclusion

The presented data revealed that projects using primarily mature technologies, especially the C and C++ programming languages have a lower adaption rate of CI/CD, but other characteristics of CI/CD pipeline usage are almost identical to general observations. Approximately 30% of C/C++ projects are using CI/CD pipeline(s) in their workflows, which is less than the values measured by researchers in terms of newer and/or more popular programming languages.

This research shows that, the vast majority of projects hosted on GitHub use GitHub Actions as CI/CD provider. In future research, it would be worthwhile to include other VCS platforms into the data collection, to get more complete results from the characteristics of CI/CD usage.

It have been found that CI/CD usage rate is higher in the case of older projects than in the case of projects created less than 1 year ago.

It was also observed that, in terms of C/C++ codebases, the popularity of the project affects the rate of CI/CD usage. More popular projects use CI/CD pipelines in greater proportion.

Only open-source projects are examined in this study, so the results cannot be generalized to all C/C++ projects. In the case of enterprise proprietary software, the proportion of on-premise CI/CD solutions is likely to be higher than managed CI/CD services.

In addition to the CI/CD systems, the output of several development-supporting services (e.g. static code analyzers and vulnerability scanners) are appeared in the collected raw data. Examining output of these is an interesting future research opportunity.

References

[1] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., and Filkov, V.: Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, 2015, URL https://doi.org/10.1145/2786805.2786850.

[2] Hilton, M., Tunnell, T., Huang, K., Marinov, D., and Dig, D.: Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ACM, 2016, URL https://doi.org/10.1145/2970276.2970358.

[3] Beller, M., Gousios, G., and Zaidman, A.: Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, IEEE, 2017, URL https://doi.org/10.1109/msr.2017.62.

[4] Durieux, T., Abreu, R., Monperrus, M., Bissyande, T. F., and Cruz, L.: An analysis of 35+ million jobs of travis CI. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2019, URL https://doi.org/10.1109/icsme.2019.00044.

[5] Chen, L.: Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, **32**(2), (2015), 50–54, URL https://doi.org/10.1109/ms.2015.27.

[6] Shahin, M., Babar, M. A., and Zhu, L.: Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, **5**, (2017), 3909–3943, URL https://doi.org/10.1109/access.2017.2685629.

[7] Dingare, P. P.: *CI/CD Pipeline Using Jenkins Unleashed*. Apress, 2022, URL https://doi.org/10.1007/978-1-4842-7508-5.

[8] Felidre, W., Furtado, L., da Costa, D. A., Cartaxo, B., and Pinto, G.: Continuous integration theater. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, 2019, URL https://doi.org/10.1109/esem.2019.8870152.

[9] Henschel, J.: A comparison study of managed CI/CD solutions. 2020, URL https://git.cubieserver.de/jh/cs-e4000-seminar.

[10] Kerzazi, N. and Adams, B.: Who needs release and devops engineers, and why? In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, ACM, 2016, URL https://doi.org/10.1145/2896941.2896957.

[11] Gruhn, V., Hannebauer, C., and John, C.: Security of public continuous integration services. In *Proceedings of the 9th International Symposium on Open Collaboration*, ACM, 2013, URL https://doi.org/10.1145/2491055.2491070.

[12] GitHub Actions: *Usage, limits, billing and administration*. URL https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration. Retrieved 2022-09-01.

[13] Travis CI: *Pricing*. URL https://www.travis-ci.com/pricing/. Retrieved 2022-09-01.

[14] Circle CI: *Pricing*. URL https://circleci.com/pricing/. Retrieved 2022-09-01.

[15] GitHub: *Commit statuses - GitHub Docs.* URL https://docs.github.com/en/rest/commits/statuses. Retrieved 2022-09-05.