Accepted for publication in Empirical Software Engineering
(to appear in 2023)

On the usage, co-usage and migration of CI/CD tools: a qualitative analysis

Pooya Rostami Mazrae · Tom Mens · Mehdi Golzadeh · Alexandre Decan

Preprint – January 2023

Abstract Continuous integration, delivery and deployment (CI/CD) is used to support the collaborative software development process. CI/CD tools automate a wide range of activities in the development workflow such as testing, linting, updating dependencies, creating and deploying releases, and so on. Previous quantitative studies have revealed important changes in the landscape of CI/CD usage, with the increasing popularity of cloud-based services, and many software projects migrating to other CI/CD tools. In order to understand the reasons behind these changes in CI/CD usage, this paper presents a qualitative study based on in-depth interviews with 22 experienced software practitioners reporting on their usage, co-usage and migration of 31 different CI/CD tools. Following an inductive and deductive coding process, we analyse the interviews and found a high amount of competition between CI/CD tools. We observe multiple reasons for co-using different CI/CD tools within the same project, and we identify the main reasons and detractors for migrating to different alternatives. Among all reported migrations, we observe a clear trend of migrations away from Travis and migrations towards GitHub Actions and we identify the main reasons behind them.

Keywords CI/CD · Collaborative Software Development · Workflow Automation · Qualitative Analysis · Empirical Software Engineering

P. Rostami Mazrae, T. Mens and M. Golzadeh
Software Engineering Lab, Université de Mons, Mons, Belgium
E-mail: firstname.lastname@umons.ac.be

A. Decan (F.R.S.-FNRS Research Associate)
Software Engineering Lab, Université de Mons, Mons, Belgium
E-mail: alexandre.decan@umons.ac.be

1 Introduction

Continuous integration and deployment (CI/CD) is considered a crucial practice to support collaborative software development [1, 2]. CI/CD gained its popularity as a software engineering practice thanks to the eXtreme Programming methodology introduced by Beck et al. [3]. CI/CD helps to automate a wide range of activities during software production, including compilation, building, testing, quality assurance, dependency and security management, creating releases, and many more [4–7]. As a result, CI/CD helps to produce higher quality software releases at a faster pace and with less effort [8]. This has led CI/CD (that we will henceforth abbreviate to CI) to become one of the most important collaborative software development practices for companies and open source software (OSS) communities worldwide. Its use ensures integrity and control over all changes made to the software project [9, 10].

There is a wide range of CI tools to help developers automate their development workflow. Popular examples of contemporary CI tools are GitHub Actions, GitLab CI/CD, Azure DevOps, CircleCI, Jenkins, and Travis. Each of them has its own benefits to accommodate the specific needs and constraints of individual software projects [10].

Recently, the landscape of CI tools has witnessed important changes due to the emergence of new competing tools, support for more operating systems in existing CIs, changes in billing policies, changes in the company or community structure of the CI tool provider, reliability and performance of the CI services being provided, and many more [10]. In particular, the introduction in November 2019 of GitHub Actions (that we will henceforth abbreviate to GHA) as a fully integrated CI service on GitHub has led both new and existing GitHub repositories to adapt or migrate to this service as their primary CI tool [11, 12]. At the same time, Travis has exhibited a progressive decrease in popularity these recent years, due to a combination of quality of service problems and restrictions imposed on its free plan for OSS projects [11]. In light of these important recent changes in the CI landscape, this article has two main research goals:

G_1: As the first goal, we aim to understand the rationale behind how and why experienced developers in commercial and OSS projects rely on specific CI tools, and how this usage has changed in comparison with previously reported research studies. This goal is broken down in five specific research questions:

$RQ_{1.1}$ Which CI tools are being used?
$RQ_{1.2}$ What are the main reported reasons for using CI?
$RQ_{1.3}$ Which activities are being automated by CI tools?
$RQ_{1.4}$ What are the most valuable features of these CI tools?
$RQ_{1.5}$ What are the reported shortcomings of these CI tools?

G_2: As the second goal, we aim to understand the reasons for co-using different CI tools simultaneously, as well as the reasons for migrating from one CI tool to another. This goal is motivated by the recent changes in the CI landscape on GitHub, the most popular software development hosting platform for OSS projects today. Less than 18 months after its introduction [11], GHA became the most widely used CI tool on GitHub, taking over Travis that has been available and dominant for years on GitHub.

$RQ_{2.1}$ Why are multiple CI tools co-used simultaneously?
$RQ_{2.2}$ Why do software projects migrate to a different CI tool?
$RQ_{2.3}$ What are the difficulties in carrying out a CI migration?

In order to achieve these goals and to provide answers to the eight research questions, we carried out a qualitative analysis by conducting in-depth interviews with 22 experienced software practitioners.

The remainder of this paper is organised as follows. Section 2 presents the related work of earlier empirical studies related to CI usage. Section 3 introduces the design, setup, and process of the qualitative study we conducted. Based on the insights obtained from the interviews, Section 4 answers the research questions for goal G_1, while Section 5 focuses those for goal G_2. Section 6 discusses some additional insights related to our findings. Section 7 presents the threats to validity of the conducted study. Finally, Section 8 concludes.

2 Related work

2.1 CI/CD usage practices

Probably the best entry point to CI/CD usage practices are the systematic literature reviews (SLR) by Shahin et al. [2] and Soares et al. [10].

The SLR of Shahin et al. [2] covered 69 scientific articles published up to 2016. Its aim was to synthesize the reported approaches, tools, challenges, and practices for adopting and implementing continuous practices. The included studies mostly show an increase in adoption of continuous practices, and discuss the integration problems faced by projects trying to use these practices. They also report that teams mostly use continuous practices to reduce build and test time, increase the visibility and awareness on build and test results, detect violations, flaws, and faults, and improve the deployment pipeline w.r.t. security, scalability, dependability and reliability. While the authors identified 30 approaches and associated tools, many of the CI tools in our own analysis (e.g., GHA, Travis, CircleCI, Azure DevOps, AppVeyor) were not even mentioned by this SLR. One of the reasons for this was that the SLR was considerably broader in scope than our study, including also tools for version control, build systems, code quality analysis, testing, configuration, provisioning and deployment.

The SLR by Soares et al. [10] covers 101 scientific articles reporting on the use of CI and published prior to 2019. The SLR aimed at identifying and interpreting empirical evidence regarding how CI impacts software development. CI usage was reported to correlate with improved productivity, efficiency, and developer confidence. CI practices were observed to benefit the software process by promoting faster iterations, more stability, predictability, and transparency in the development process. CI also benefits pull-based development by improving and accelerating the integration process. The SLR concluded that most of the existing research highlighted the positive effects of CI usage, leaving room to study the challenges and shortcomings of using CI tools. Since this SLR did not consider any publications after 2019, it did not include any study reporting on the impact of GHA on the CI landscape, given that GHA was only publicly introduced as a CI service in November 2019.

The aforementioned SLRs included many publications on how CI/CD practices have been implemented in different environments in order to identify their potential benefits [5,13–16], challenges and shortcomings [6,17,18]. This reflects the importance of CI/CD practices and their impact on software development practices. In the following subsections we narrow down on case studies that explored the use of CI/CD (Section 2.2), as well as the specific use of Travis (Section 2.3) and, more recently, by GHA (Section 2.4).

2.2 Case studies on CI/CD usage

Chen [13,17] reported on the benefits and adoption challenges of CD practices in Paddy Power, a large company. Among the achieved benefits from CD adoption, he reports an accelerated time to market, the ability to build the right product, an improved productivity and efficiency, the increased reliability of releases, as well as an improved product quality and customer satisfaction.

Betz et al. [19] studied the impact of adopting a CI tool to develop AMBER, a molecular dynamics software package widely used in chemical industry. They report an improved collaboration and communication between globally distributed developers. The CI tool also enabled real-time reporting of failure and benchmark information, a task that would be time-consuming for individual developers to achieve by themselves.

Lu et al. [20] reported on a case study on D5000, a smart grid scheduling support system. The results show that using continuous integration and automated testing resolves quality and integration issues effectively and efficiently, without introducing considerable overhead.

Kulas et al. [21] reported on how CI practices helped to reduce the development time of ARGOS (Advanced Rayleigh guided Ground layer adaptive Optics System), a software designed to solve a specific problem with images produced by a telescope. Commissioning time for an instrument at an observatory is costly, especially at night. Whenever astronomers come up with a software feature request or point out a software defect, the software engineers should find, implement and deploy a solution as fast as possible. Using Jenkins to automate testing allowed the team to guarantee the correctness of the proposed changes while respecting strict time constraints.

Gmeiner et al. [22] carried out a case study on the usage of CI tools in an Austrian online business company. They highlighted the complex technical and organizational challenges based on more than six years of practical experience in establishing and maintaining an effective continuous delivery pipeline.

Savor et al. [8] carried out a mixed-method study of CI/CD usage at Facebook and OANDA. The study revealed that the CD part of the used tools could not be used to its full potential. For example, OANDA's policies prevented the company from fully embracing continuous deployment, leading to delays in delivering new features to end-users.

Elazhary et al. [16] conducted case studies with three software development organizations that implemented CI practices, in order to identify the benefits and challenges related to them. Based on interviews with 18 employees (developers, managers, team leads, and directors) they identified the following CI practices: maintaining a source repository, automating the build, automating the tests for a build, making daily commits to the mainline, ensuring that these commits build on an integration machine, having fast builds, testing in a clone of the production environment, providing a Docker executable of the latest release, ensuring visibility of the system state and changes, and automating the deployment. Based on a trace-log analysis, the authors also studied the impact of implementing these practices. Some observed good impacts were: minimizing merge conflicts, increasing consistency and reproducibility of builds, more reliable bug detection, minimizing breaking changes, better and faster developer and customer feedback, reducing build complexity and facilitating onboarding. Some observed challenges were difficulties to test the UI, longer build times due to automated tests, bottlenecks for committing due to PR reviews, scalability issues, and increased maintenance effort. This reveals that CI practices have their merits, but also bring along their own difficulties.

2.3 On the use of Travis in GitHub projects

There are many empirical studies that have studied the use of Travis, given it was the dominant CI tool on GitHub. These studies cover various different aspects such as the practices developers follow to use Travis [23], its benefits and shortcomings [24], or the observed antipatterns [25].

Vasilescu et al. [4] conducted a quantitative analysis on 246 GitHub projects to study the improvements that Travis can bring to the pull-based development process. They observed a higher volume of pull requests being accepted, and more defects being discovered thanks to Travis usage.

Hilton et al. [5] studied 34,544 OSS projects on GitHub and surveyed 442 developers to understand how developers use CI. They found out that there are still many OSS development teams that do not use a CI tool due to a lack of familiarity. However, among those that used a CI tool, 90% of them used Travis. They reported that popular projects are more likely to use CI and the median time for CI adoption is one year. They found that the use of Travis helped developers to catch bugs earlier. They also found that projects using Travis had more than twice as many releases and faster pull request integration time, while avoiding acceptance of pull requests that would break the builds.

In an explorative analysis of Travis on GitHub, Beller et al. [6] found that Travis usage had increased a lot by 2017, being used in one-third of popular projects on GitHub. Their analysis of 2.6M+ Travis builds for Java and Ruby projects revealed that Travis usage was highly focused on testing-related tasks, primarily to enable developers to test their software across different OS environments. However, the CI tool was not able to replace local testing because of the high latency (often more than 20 minutes) between writing the code and receiving feedback from the automated tests.

Gupta et al. [26] studied how the introduction of Travis impacted developer attraction and retention in 217 GitHub repositories. Contrary to their expectations, they found statistical evidence that developer attraction and retention in these projects was higher in the year before adopting Travis than in the year after.

Widder et al. [7] quantitatively studied 7,276 GitHub projects that had abandoned Travis. They observed that projects with more pull requests are less likely to abandon Travis, while projects with more commits are more likely to do so. They also observed that a project's dominant language is an important predictor for Travis abandonment. Finally, contrary to the intuition, they found that projects with more complex configurations tend to be less likely to abandon Travis. In a follow-up study, Widder et al. [27] identified the pain points of Travis as a CI tool in software development projects on GitHub. They used a combination of online surveys (132 respondents), interviews (12 respondents) and quantitative analysis using logistic regression on a dataset of 6,239 GitHub projects to predict Travis abandonment. Some of the identified pain points were unsupported technology, long build times, infrequent changes, poor user experience, and build failures.

2.4 On the use of GHA in GitHub projects

Due to its recent introduction in November 2019, there is only a limited number of studies that have focused on GHA despite its impact on the CI landscape on GitHub.

Golzadeh et al. [11] presented a longitudinal quantitative study on the use of CI tools in over 91K GitHub repositories of distributed npm packages. They observed that more and more repositories are relying on a CI tool, reaching up more than 50% of the repositories in May 2021. They found that GHA and Travis dominate the CI landscape and are used by 90% of the repositories with a CI tool. They also found that GHA took over Travis in popularity in only 18 months after its introduction, a consequence of many repositories that started to use GHA instead of Travis.

Kinsman et al. [12] analysed the impact of adopting GHA in 3,190 repositories and observed that the adoption of GHA increases the number of rejected pull requests and decreases the number of commits in merged pull requests.

Through a manual inspection of 209 issues related to GHA, they observed that developers have an overall positive perception of GHA. These observations were confirmed by Chen et al. [28] in a replication study on 6,246 repositories.

Valenzuela-Toledo and Bergel [29] investigated the use and maintenance of GHA workflows in 10 popular GitHub repositories. They manually inspected 222 commits related to workflow changes and determined 11 different types of workflow modifications. They uncovered a number of deficiencies in GHA workflow production and maintenance, calling for adequate tooling to support creating, editing, refactoring, and debugging workflow files.

Decan et al. [30] analysed the use of GHA in nearly 70K GitHub repositories in order the get a deeper insight into the GHA ecosystem. They found that 43.9% of the repositories are using GHA workflows, and they characterized these repositories and their workflows, in terms of which jobs, steps, and reusable Actions were used and how. They notably observed that workflows are primarily used for development purposes, despite the fact that many other kinds of activities could potentially be automated with GHA. They also observed that nearly all workflows rely on Actions, which may be problematic since issues in these Actions (e.g., bugs, security vulnerabilities, outdated or obsolete components) can propagate to the workflows that use them, potentially affecting the entire GHA ecosystem. They call for more in-depth empirical studies to provide a comprehensive understanding of the GHA ecosystem.

3 Methodology

The two goals of this article were defined with this related work in mind. Goal G_1 aims to understand the rationale behind how and why experienced software developers use specific CI tools, and how this usage has changed in comparison with previously reported studies. Goal G_2 aims to understand how developers co-use CI tools and why they migrate to different CI tools, especially in the light of the rapidly changing CI landscape due to the introduction of GHA on GitHub. In order to achieve these goals, we carried out a qualitative analysis by conducting semi-structured interviews with experienced software developers around the globe. The remaining of this section is structured as follows: Section 3.1 explains how we created our interview questionnaire, Section 3.2 how we selected the interview participants, and Section 3.3 how the interviews were conducted, processed, and coded.

3.1 Interview questionnaire

All co-authors of this paper jointly created an interview questionnaire aiming to capture all the aspects we wanted to cover to reach research goals G_1 and G_2. To validate the questionnaire, dry-runs were carried out with three distinct developers with experience in CI/CD. The results of the interviews with these developers were not included in our analysis, as they only served to further improve the questionnaire.

The final questionnaire is presented in Appendix A. It included about 30 questions, some being conditional to the answers to previous questions. These questions were structured along the following main themes:

1. General questions about the respondent
2. General questions about CI/CI usage
3. Questions about specific CI/CD tool usage
4. Questions about CI/CD migration
5. Questions about CI/CD tool co-usage
6. An open-ended closing question

The responses for themes 2 and 3 were used as a basis for research goal G_1 (see Section 4), while themes 4 and 5 served as a basis for research goal G_2 (see Section 5).

3.2 Selection of respondents

We targeted interview candidates with experience in software development in open source as well as in commercial settings. Our main strategy to find interview candidates was through the Twitter and LinkedIn channels of the authors, through e-mails and direct messages to practitioners, and through referrals by colleagues as well as by some interviewees. To increase diversity of interviewees and not being restricted by geographical constraints, we decided to conduct our in-depth interviews online using video conferencing tools.

In order to be able to participate in the study, candidates needed to meet at least two out of three inclusion criteria that we have defined beforehand: (1) having actively contributed to, or having been responsible for a software project relying on CI; (2) having sufficient knowledge about the reasoning and decision-making process about which CI tool is used in that software project and how; (3) having been involved in setting up or maintaining the CI process of the project.

We stopped selecting and interviewing candidates when we reached a point of saturation [31, 32] where no new themes or codes emerged from the additional data collected. We observed such saturation after the 20th respondent when, except for the answers to the open-ended closing question and the specific work context of the respondents, little additional relevant information was gathered on top of what previous respondents had already provided. We therefore stopped the interview process after the 22nd interview. While only 22 interviews might seem little, it is more than what has been used in some previous qualitative studies in empirical software engineering [33] [34] [35] that reported saturation after 16, 16 and 10 interviews, respectively. Nevertheless, we acknowledge that our inclusion criteria for selecting interview candidates were such that we only considered experienced developers with practical expertise in CI usage. As a result, the opinions and findings reported in the paper do not necessarily generalise to more inexperienced developers.

Table 1 summarises the demographics of the respondents. In the remainder of this article, the respondents are identified by a unique number R_n or

Table 1 Characteristics and demographics of respondents.

ID	years of experience dev.	CI	industry	open source	continent
R_1	7	-	✓	✓	Europe
R_2	11	-	✓		Europe
R_3	6	-	✓		Europe
R_4	19	-	✓	✓	North America
R_5	22	14	✓	✓	Europe
R_6	19	-	✓		North America
R_7	8	-	✓		Europe
R_8	11	8	✓	✓	Europe
R_9	6	4.5		✓	Europe
R_{10}	20	10	✓	✓	North America
R_{11}	5	4	✓	✓	Europe
R_{12}	8	6	✓	✓	Europe
R_{13}	15	-		✓	Europe
R_{14}	4.5	3	✓	✓	Asia
R_{15}	10	3	✓		Europe
R_{16}	12	2	✓		Asia
R_{17}	15	-	✓		Europe
R_{18}	10	-	✓	✓	Europe
R_{19}	24	-	✓	✓	Europe
R_{20}	15	4	✓	✓	Europe
R_{21}	12	8		✓	North America
R_{22}	20	12	✓	✓	Europe

simply n when it is clear from the context. The second and third columns of the table report on the number of years of development and CI experience of each respondent. On average, the respondents can be considered as very experienced software developers, with an average of 12 years and 4 months of software development experience and 4.5 years of CI experience. Not all respondents dissociated their years of CI experience from their years of development experience, explaining the absence of the second number for some respondents.

Columns 4 and 5 of the table reveal that respondents were involved in a wide range of software development projects, including personal, open source software (OSS) projects and commercial projects. Most of the respondents (12 out of 22 respondents) had both industrial and open source software experience, while seven respondents had only been involved in commercial software, and three of them were only in OSS. Furthermore, some of the respondents were or had been working on big open source projects like curl and Conda-forge, or for big tech companies such as LinkedIn and Microsoft. The last column reveals that most of the respondents lived and worked in Europe (16 respondents spread over 7 different Western European countries), while 4 respondents came from North America and 2 from Asia.

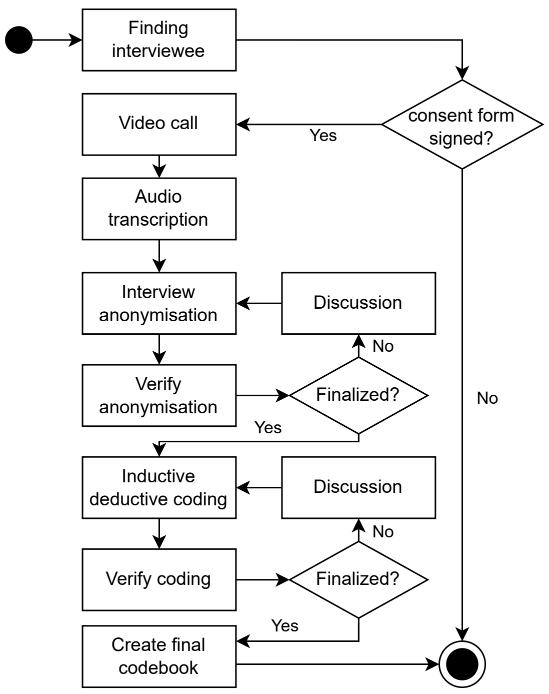

Fig. 1 Schema of the interview process.

3.3 Conducting and processing the interviews

The process we followed for conducting and processing each interview is summarised in Figure 1. Prior to each interview, the selected interview candidate was required to sign a consent form in order to meet the GDPR regulations and to allow us to use the interview results for research purposes. After having received the consent form, a virtual meeting was fixed to carry out the online interview through a video-conferencing tool the candidate was comfortable with. One author conducted the interview and made an audio recording, with the explicit permission of the candidate. Each of the 22 interviews lasted roughly about 30 to 45 minutes, and the total set of interviews was spread over a four-month period, from November 2021 to February 2022.

The author that conducted the interview resorted to an automatic transcription tool to transcribe each interview. The resulting verbatim textual transcripts were cleaned and anonymised to hide privacy-sensitive information such as names of persons, companies, or specific software projects. This process was made by one author, and was checked and further improved by a second one. A third author was involved in case of doubt.

To structure the information gained from the interview transcripts we followed a process similar to Foundjem et al. [33], using a combination of *inductive* and *deductive coding* [36]. In the first phase of inductive coding, the first author assigned labels to the transcribed text, without any predetermined theory, structure, or hypothesis. After that, one author followed a top-down deductive coding process to create separate codebooks for each interview, deriving codes based on the research questions and concepts under study, and using these codes to group and structure the inductive labels that were attached to the transcribed text during the inductive coding phase. A second author verified each of these codebooks and, in case of disagreement, a third author was involved in the discussion until we reach a consensus on the coding.

All anonymised transcripts except two have been made available as supplementary material to this paper. We did not receive authorisation from the respondents of the two excluded transcripts to make this information public, although we were authorised to use and process the information from those transcripts in the context of this paper.

4 Goal G_1: Why, how and which CI tools are being used?

This section addresses our first research goal, aiming to understand the rationale behind how and why developers rely on specific CI tools, and how this reported usage has changed in comparison with the existing body of research presented in Section 2. We will do so by providing answers to the following research questions:

$RQ_{1.1}$ Which CI tools are being used?
$RQ_{1.2}$ What are the main reported reasons for using CI?
$RQ_{1.3}$ Which activities are being automated by CI tools?
$RQ_{1.4}$ What are the most valuable features of these CI tools?
$RQ_{1.5}$ What are the reported shortcomings of these CI tools?

These research questions will be addressed in the next five subsections.

$RQ_{1.1}$ Which CI tools are being used?

This preliminary research question aims to reveal the diversity of CI tools being used by respondents, and to determine which CI tools have been used more frequently by respondents. Overall, 31 different CI tools have been reported by respondents. The full list of reported CI tools can be found in Appendix B. Throughout the article we use respondent IDs whenever we cite relevant quotes from them. In order to put these quotes in the right perspective, Appendix B also provides a mapping between these IDs and the CI tools they reported having used.

Table 4 lists the 14 CI tools that were used by at least two different respondents at some point in time, ordered in decreasing frequency of usage. One can observe the use of a large variety of CIs, some of them being self-hosted (e.g., Hudson, Jenkins), others being offered as a cloud service (e.g., GHA,

Table 2 CI tools having been or being used by at least 2 respondents.

	cloud	**self**	**open**	**release**	**# of respondents**	
CI tool	**based**	**hosted**	**source**	**date**	**all-time**	**currently**
GHA	✓			Nov 2019	18	18
Jenkins		✓	✓	Feb 2011	16	9
Travis	✓			Nov 2011	15	1
GitLab CI/CD	✓	✓	✓	Nov 2012	14	12
CircleCI	✓	✓		Sep 2011	12	8
Azure DevOps	✓	✓		Oct 2018	11	9
AppVeyor	✓			Nov 2011	5	3
Hudson		✓	✞	Feb 2005	5	0
TeamCity	✓	✓		Oct 2016	3	3
Cruise Control		✓	†	Mar 2001	2	0
Drone	✓	✓	✓	2014	2	2
Bitbucket Pipelines	✓			May 2016	2	2
Netlify	✓	✓	✓	Apr 2015	2	2
Bamboo		✓		Feb 2007	2	1

† Cruise Control was not open source when it was being commercialised by ThoughtWorks. The tool became open source after the company stopped maintaining it.
✞ Hudson was originally released as open source by Sun, but was commercialised when Oracle acquired Sun.

Travis, Bitbucket Pipelines) or both (e.g., GitLab CI/CD). In addition to the 14 CI tools listed in Table 4, another 14 CI tools were reported only once. These were, in alphabetical order: AWS CI/CD, Buildbot, BuildKite, Cirrus CI, Codefresh, Concourse, Heroku, Jacamar CI, Percy, Pulumi, Sauce Labs, Tekton, Vercel, and Zuul. Three respondents additionally reported resorting to *custom-built in-house* CI solutions, since no existing CI tool satisfied all of their needs. These solutions will not be considered in this paper.

Table 4 also summarises which of the reported CI tools are still being used *currently* by the respondents. In the light of the second research goal, we observe that GHA is the most frequently reported CI tool one by far, with the large majority of respondents (18 out of 22) using it currently. The opposite can be observed for Travis: nearly all respondents that were using it at some point in time are no longer using it, despite Travis being still actively maintained. For instance, only 1 of the 15 respondents having used Travis is still using it these days. This corroborates the results of Golzadeh et al. [11] on the popularity of GHA at the expense of Travis.

Jenkins falls in between GHA and Travis. It used to be a popular self-hosted CI since the majority of respondents (16 out of 22) reported having used this CI at some point during their software development experience. Yet, only 9 out of them are still using Jenkins. The reasons why such changes occurred will be explored later in this article.

GitLab CI/CD, CircleCI and Azure DevOps are three other popular CI tools, having been used by at least half of the respondents, and still be used by most of them. On the other hand, none of the respondents are currently using Hudson nor Cruise Control, two of the earliest commercial self-hosted CI tools. The reason is that both Hudson and Cruise Control were discontinued by their respective companies and replaced by a new CI tool. For instance, Thoughtworks, the company owning Cruise Control, replaced it with a new commercial CI tool named Cruise in 2010. Since Cruise was not based on Cruise Control, the company decided to make the source code of Cruise Control publicly available after discontinuing its support. A few years later, Thoughtworks rebranded and renamed Cruise as GoCD, which was ultimately released as an open source CI tool in 2014. Hudson, the open source Java-based CI tool, used to belong to Sun Microsystems, until Oracle decided to acquire this company and to commercialise Hudson. The open source community reacted by creating Jenkins, an open source fork that became much more popular than its ancestor. Jenkins continued to grow and to increase its functionalities, while Hudson stagnated and ultimately became discontinued in February 2017.

31 distinct CI tools have been reported by the respondents, of which 14 are used by at least 2 respondents. Some CI tools, such as GHA, Jenkins, Travis, GitLab CI/CD, CircleCI and Azure DevOps were used by at least half of the respondents at some point in time. Only GHA and GitLab CI/CD are currently in this situation. While Travis and Jenkins were among the most used CI tools, most respondents have stopped using Travis and, to a lower extent, Jenkins.

$RQ_{1.2}$ What are the main reported reasons for using CI?

This research question aims to identify the reasons behind adopting CI in software development projects. Based on a survey with several hundreds of developers, combined with interviews with 16 developers from 14 different companies, Hilton et al. [5, 37] studied, among other aspects, the developer's motivations and benefits of using CIs. They reported that developers use CI for 8 different reasons: to help catch bugs earlier; to avoid breaking builds; to provide a common build environment; to deploy more often; to allow faster iterations; to make integration easier; to enforce a specific workflow; and to allow testing across multiple platforms. Many other qualitative studies have reported similar reasons for using CIs [1,4,14,15,38–40]. The SLR [10] mentioned the following reasons: improved software quality, stability, predictability, and transparency; faster build, integration, and release cycles; improved productivity, efficiency, and developer confidence; reduced workload; and faster detection and resolution of defects. Our interview results align with these reasons since respondents reported adopting CI to achieve the following goals:

– increased reliability: "*the intent was to ensure that we had reliable outputs. Whenever there's a code change, we would know that it's working.*" [R_{17}] and "*Basically there were plenty of people contributing, so [for] each pull request [we] needed to make sure that this pull request was not breaking the code and that the code was reaching production.*" [R_{11}]

– increased quality (e.g., through better reproducibility of bugs, increased testing, and performing quality checks): "*The reason was quality, we wanted to use CI/CD to run tests all the time and to deploy automatically to package automatically the software without depending on one person to do it. So the goal was really to have a common view on the build process of the tools and to improve the quality.*" [R_{13}]
– increased productivity: "*We very much invested into it that, if people want to contribute, they can really focus on the actual contribution, and there is very little overhead for them or fellow maintainers to do. We try to automate as much as possible.*" [R_{10}]
– faster delivery: "*The idea was to deliver value to the company in a quick time.*" [R_{11}]
– reduced cost and effort: "*The human costs have been reduced over time because of all the automation that arrived in those tools.*" [R_{13}] and "*[...] checked by the linting and [...] the maintainer does not need to do that extra work.*" [R_{13}]
– rapid feedback: "*[...] being able to have a quick feedback, it's also why we work in parallel. We work to reduce the time of the pipeline so we can get early feedback for the people that contribute changes.*" [R_{13}]
– increased transparency of the build process: "*When we publish a release, people can check from which CI tool it is coming. They can see the logs of the build. They can see that the build has not been tampered.*" [R_{13}]

While earlier research [37] revealed that developers consider security as a barrier for using CI, many of the respondents we interviewed actually mentioned *increasing security* by reducing security vulnerabilities as an important goal for CI automation in their projects. For example, R_1 mentioned that "*on a more recent project we start to use Snyk, which is a tool for detecting vulnerabilities*". R_2 referred to the importance of *DevSecOps* which is an approach to automation and platform design that integrates security as a shared responsibility throughout the entire development lifecycle: "*something that's taking off right now is the addition of security in DevOps. It's something called DevSecOps where in the DevOps pipeline we had static code analysis, or even dynamic code analysis and that's one thing that is moving into DevOps area, which is how to integrate security operations and development.*" R_{14} and R_{15} highlight the importance of security testing/scanning: "*in my current organization we have automated security tests in our repo so when we want to deploy something in the production [...] we configured that thing to automatically test our application security and give us a report of what we need to fix and if the security tool fails*" [R_{14}], and "*We also have some scanning tools which we facilitate as a security scan to check the vulnerabilities or such things*" [R_{15}].

R_{10} additionally reported that using a CI enabled their open source project to retain contributors and attract new ones since CI usage allows to reduce the maintenance overhead: "*We invested a lot of thought and time into how to attract and retain contributors. [...] we thought about how we can lower the barrier to contribute, but also remove as much overhead as possible based on the* *assumptions that as an open source project you really have to be fun, otherwise people will move on to other projects.*" It is interesting to note that this positive impact of CI usage on contributor attraction and retention was *not* confirmed in an empirical study by Gupta et al. [26] on 217 GitHub repositories using Travis. Surprisingly, they statistically observed that developer attraction and retention of a project were higher in the year *before* adopting Travis than in the year *following* Travis adoption. More research would be needed to ascertain the relationship between CI usage and contributor attraction and retention.

R_8 reported the cloud-based nature as a reason for using Travis: it allowed the team to reduce cost and hardware resources, since they were able to use the CI tools "as a service" compared to many other competing CI tools at that time that mostly required self-hosting.

The main reported reasons for CI adoption are to increase reliability, productivity and security, to improve speed of delivery, and to reduce cost and human effort.

The reasons reported by interview respondents are in line with earlier findings in the scientific literature. For instance, they were reported in the SLR by Soares et al. [10] and Elazhary et al. [16]. Savor et al. [8] also reported that CI tools allow software development companies to increase their team size by a factor of 20 and their code base by a factor of 50 without decreasing developer productivity or software quality. These findings are confirmed by our respondents who argued that CI usage increases reliability, quality, and productivity. On the other side of the coin, interview respondents highlighted that adopting CI tools introduces an additional layer of complexity into the development environment which needs to be carefully considered.

$RQ_{1.3}$ Which activities are being automated by CI tools?

$RQ_{1.2}$ revealed that CI automation is used for different reasons. As a consequence, one may expect that CI tools are used for a variety of activities, such as building or testing code, managing dependencies, etc. Vassallo et al. [41] even suggested continuous refactoring as an additional activity to automate by CI tools to control the complexity of software changes.

Table 3 reports on the activities that respondents reported for automation as part of their CI tool usage, distinguishing between the activities that were initially automated as part of the CI process, and the ones that were automated later on. As can be seen from the second column of Table 3, *build automation* is unsurprisingly the most popular activity (mentioned by 9 respondents) that is initially part of a CI process. An equally popular activity (also mentioned by 9 respondents) is *unit testing*. Respondents also mentioned other testing-related activities during the initial phase of CI usage, namely *code coverage analysis* (4 respondents), *integration testing* (3 respondents), and *end-to-end testing* (2 respondents). As an example of how respondents use CI tools for testing, R_4

mentions: "*we merged 600 pull requests from 170 people and I'm not going to run the tests manually, there's no way to scale a project like that unless you have automation behind the testing*". Another 3 respondents report automated *code quality analysis* as one of their initial reasons for using CI tools. For instance, "*I use a lot of other stuff like linting, automated code formatting, coverage*" [R_{10}]. Other reasons for initially using CI tools were *generating documentation*, *server provisioning*, *checking browser compatibility*, and *creating multiple builds* (e.g., "*we are also using an environment variable on GitLab CI to use different configs for each project*" [R_{16}]). Furthermore, R_9 reports using CI tools to make sure open source contributions are not breaking any previous functionality in the program. This respondent emphasises that "*it is especially important to do that for cross-platform programs, because developers usually only work on one system.*"

Column three of Table 3 reports on those activities that had been added later on to the CI automation process. The most popular of those activities was *security analysis*, being reported by 8 respondents. The *packaging and deployment* phases of the CI/CD process also tend to be added in a later phase (7 respondents). The same holds for *code quality analysis*, mentioned by 7 respondents as being added later on to the CI automation (as opposed to 3 respondents that started automating it in the initial phase of using a CI tool). More advanced testing activities (beyond unit testing) were also reported more frequently to be added later on.

Other activities that were reported to be added later on to the CI automation were *checking non-code artifacts* (7 respondents), *dependency management* (5 respondents), *license verification* (3 respondents), and *integration with communication channels* (2 respondents). For instance, R_{16} integrated the CI tool with a Slack communication channel: "*I also integrated our CI/CD with Slack. After the build is successful and the APK is generated successfully, we upload the APK to different channel of Slack for our customer or testers*". R_{21} uses this integration to learn about forced pushes: "*We have one that notifies our Slack channel when someone forced pushes*".

Related to *checking non-code artifacts*, R_{13} reports checking the format of commits to be the same as the expected commit format. R_{16}, R_{20}, and R_{22} reported using linting tools for checking non-code related artifacts. Additionally, R_{18} reported, "*doing style checks, formatting checks of the files, and also typing checks*". R_{19} indicated "*in addition to these CI services that run our particular jobs, there's also these services that do, for example, code analysis that may be not exactly CI services, but they are services that run and do things on the code based on commits. Maybe they would qualify as CI services. [...] That's sort of a popular thing these days, for example, to do a static code analyzer service*".

The following activities were mentioned by only one respondent to be added later on to software automation:

– *software verification*: "*as a team grows to 10, 15, 20, 40 people, now, it becomes a place to introduce constraints and system checks and verifications*

Table 3 Activities being automated by CI tools

activity	**initially**	**added later**
build automation	9 12 13 14 15 16 17 18 19	–
unit testing	1 2 3 4 5 6 7 9 19	13
integration testing	1 6 8	5 11 18 19
end-to-end testing	3 6	18
code coverage analysis	1 4 5 10	17 19
code quality analysis	7 10 13	2 6 15 17 19 21 22
security analysis	–	1 2 4 13 14 15 17 22
packaging and deployment (CD)	–	6 9 11 13 14 18 21
checking non-code artifacts	–	13 16 18 19 20 21 22
dependency management	–	8 13 19 21 22
integrating with comm. channels	–	16 21
license verification	–	17 21 22
other	1 2 16 20	6 21

that go beyond what you could do from tribal knowledge. So now it almost becomes a system where you take the guidelines that you would write down in a document and you put them into automation." [R_6],

- *labelling/closing pull requests*: "*We have a GitHub action that labels pull requests with the appropriate labels*" [R_{21}] and "*Someone just added one in the last week that closes stale pull requests*" [R_{21}],
- *detecting inactive contributors*: "*So I wrote a couple of scripts that are run via GitHub action. [...] It finds anybody who hasn't landed or reviewed a commit in the last 18 months and flag them as a collaborator that should probably be removed and opens a pull request to remove them.*" [R_{21}].

CI tools are initially used for basic CI/CD tasks like build automation, automated unit testing and code coverage analysis. More advanced activities are added later on to the CI automation, such as more advanced testing activities, security analysis, code quality analysis, dependency management, packaging, and deployment.

These insights align with the findings in the research literature. For example, Soares et al. [10] report that CI automates boring repetitive tasks such as basic automated building, testing and deployment. Our findings also confirm the initial reasons for CI usage reported by Savor et al. [8] who studied the usage of CI tools in two different companies, as well as the initial reasons reported by Elazhary et al. [16] who studied the use of CI tools in three different organisations. In addition, they report that more complicated automation tasks tend to be added later on, such as testing in a clone of the production environment. Specifically in the context of GHA, [12] and [30] report that many of the reusable Actions support the basic CI activities of building, testing and deploying.

Table 4 Most valuable features of CI tools as reported by respondents.

valuable features	GHA	Travis	Jenkins	Azure DevOps	GitLab CI/CD	CircleCI	Drone	Hudson	TeamCity	AppVeyor
good integration with hosting platform	1 2 5 6 9 10 11 13 14 16 17 18 19 21 22	10 13 19 22		12 14 15 17	1 2 3 5 11 13 16	13				
ease of use	1 2 10 11 16 17 21 22	8 22	1 2 8 22	14 15 17 18	16 22	13	9 22	15	1	
support for specific architectures/OS	9 14 19 22	14	16 18 21	9 12 19	9	12 13 19 22	9			9 13 18 19
popularity / familiarity	9 10 16 18 22	2 8 10 11 12 13 19 21 22	1 8 11 16 17 18	14				15		
good free tier	2 5 9 10 13 14 17 21 22	5 8 9 13 16 21 22		9	3 5 11					
good plugin support	6 11 13 15 21 22	8	2 13 14 17	14 15 17						
self-hosting ability	6		16 21		2 4 14		9	22	1	
useful features	5 9		11	9 15	13					
customizability	5 11 19 22		14 17							
security	18	8				18				
speed	5 11 22					13				

$RQ_{1.4}$ What are the most valuable features of CI tools?

Research goal G_1 of this article aims to understand why project maintainers rely on specific CI tools. In other words, we are interested in knowing the most valuable features offered by the CI tools that have been used by respondents, as these features are likely to play an important role in why these CI tools have been used in their projects. We therefore asked each respondent what were the most valuable features of the CI tools they had used.

Table 4 already revealed a difference between self-hosted and cloud-based CI tools, and between open source and commercial solutions. These differences may have played a role in the choice of CI tools by some developers.

Table 4 summarises the most valuable features of each CI tool, as reported by the respondents that use them. Only tools for which at least two valuable features had been reported are listed. It is worth noting that these features have to be interpreted in their historical context: they do not necessarily reflect what are the *current* valuable features offered by a CI tool, but they reflect what were these features when the respondents used the CI tool. Due to the qualitative nature of our analysis, the table is inevitably incomplete, since the absence of a respondent mentioning a valuable feature does not imply that the feature is absent from the CI tool. As a consequence of this, the valuable features of less popular CI tools are less likely to be mentioned, simply because there were fewer respondents to report about them. Below we report on the valuable features listed in Table 4.

Good integration with hosting platform. Many projects use a CI tool on top of a hosting platform (such as GitHub, GitLab, BitBucket or Azure) that is used to store and manage the project development history. In those cases, it is important for the CI tool to be well integrated into the hosting platform in order to make it as easy as possible to configure and use the CI tool. Table 4 shows that respondents appreciated the good integration of GitLab CI/CD into GitLab, the good integration of Travis and GHA into GitHub, and the good integration of Azure DevOps into Azure.

Ease of use. A good user experience makes the use of CI tools easier, smoother and more enjoyable. As such, it was considered by many respondents as one of the most valuable features of the CI tool they were using. They mentioned a variety of factors that affected the ease of use of CI tools. One such factor was the simplicity of the user interface. In addition, the presence of good documentation was also important, since it helps developers find and use the available features to their full potential. Yet another one was the ease of configuring the CI tool or the workflows or pipelines created with it, for example by providing the ability to use default settings for configurations. The variety, clarity, and above all, the stability of the available configuration options also affected the ease of use.

Support for specific architectures and/or operating systems. CI tools enable building and deploying software in specific environments, and facilitate the deployment on multiple environments. These environments typically include a particular operating system (e.g., Ubuntu or some other Linux variant, macOS, Windows, Solaris, FreeBSD) and a specific hardware architecture or processor (e.g., Intel or ARM CPUs, and AMD GPUs). Since the required environments may strongly vary from one project to another, and since not every CI tool supports all possible environments, this may affect the choice of a CI tool. For example, the ability to support Windows builds was the main reason for four respondents to use AppVeyor at the time when most of the other CI tools did not provide any (or any decent) Windows support. CircleCI was also particularly appreciated by respondents for its support for a wide variety of different build environments.

Popularity and familiarity. Several respondents reported using specific CI tools in their project out of *familiarity.* Often, developers just continue to use tools that have already been in place in the project based on some earlier decisions by former project maintainers. A related frequent reason to prefer some CI tool over another one is because of its *popularity.* If some tool is more popular than another one, it becomes more likely that it will be found or recommended by someone. Popularity was one of the main reasons raised by respondents to choose Travis or Jenkins. Until GHA entered the landscape, Travis remained the default choice for software projects hosted on GitHub, while Jenkins used to be the default choice for Java projects.

Good free tier. Most CI tools are commercial, requiring their customers to pay for the services they offer. On the other hand, many CI tools also provide what is called a *free tier* or *free plan* of their cloud-based service. This allows projects (mostly open source projects) to use the cloud resources to run the CI for free. Depending on the CI tool, the free tier may impose limitations on the number of supported users/projects, the number of minutes to execute builds, the number of monthly builds, the computing resources, type of OS, and/or the number of jobs that can be executed in parallel. Sometimes, the free tier also restricts the available functionalities of the CI. Table 4 shows that respondents particularly appreciated the free tier offered by Travis, GitLab CI/CD and GHA. As will be discussed later, the restrictions imposed on the free tier may change over time and cause projects to migrate to other CI (cf. $RQ_{2.2}$).

Good plugin support. Many respondents found it valuable that several CI tools come with the possibility to create and use reusable components for creating CI workflows or pipelines. For example, GHA distributes a large set of Actions on the GitHub Marketplace, CircleCI comes with a public registry of reusable Orbs, and Jenkins provides a large index of community contributed plugins. The amount, quality and availability of these reusable components determine to which extent a CI tool can be considered to feature good plugin support.

Self-hosting ability. As can be seen from Table 4, some CI tools can be self-hosted and be used "on premises" without needing to resort to any cloud-based service. Some companies prefer to use a self-hosted CI solution because it offers increased security, since it reduces the risk of company-sensitive information getting exposed or even compromised through cloud-based solutions: "*You can run self-hosted runners, which is a way for you to run on your own machines, but then you need to implement a whole ecosystem of security constraints because you can potentially be running arbitrary third-party code in your data center, so you need to make sure that you'll lock down that environment to make sure that the environment itself is actually secure. That's a significant investment.*"

Useful features. Many respondents reported useful features related to their CI tool of choice. For example, R_5 reported being happy with GHA since it included many useful features since its beginning, and more features are added regularly to continue making it a better tool. R_9 appreciated GHA's artifact upload support: "*they give you like 5 gigabytes or something of storage. And your CI run can upload some files. And as an administrator of that CI pipeline, you can access that file and download it like a compilation output or something.*" For the same reason, R_9 appreciated Azure DevOps. R_{15} liked the access to deployment history in Azure DevOps: "*you had a facility when you wanted to go back in time and just deploy one release that you had, for example, one year ago. You could go to the history and just click on the history and redeploy that*". R_{11} appreciated Jenkins' versatility: "*it's really powerful,* *and you can do plenty of things.*" R_{13} liked GitLab CI/CD's ability for each repository to have its own pipeline, combined with the concept of cross-project CI with multi-project pipelines.

Customizability. Several respondents considered customizability as a valuable feature, even if the interpretation of this concept varied a lot depending on the considered CI tool and respondent. The customizability of GHA was mostly referred to as the ability to use this tool for non-CI related stuff like updating the Slack channel based on the results of the runs. Respondent R_{22} even claimed that GHA had "*more options of customization*" in comparison with other CI tools. Two respondents reflected on the customizability of Jenkins, appreciating its ability to customize the user interface with different themes.

Speed. Fast building and running times were mostly valued for GHA (three different respondents). Respondent R_{13} particularly valued the speed of CircleCI due to its facility for creating complex parallel pipelines. The ability to run multiple pipelines in parallel can lead to significant speed improvements.

Security. Security aspects are of crucial importance for CI tools since the automation task they support has the potential of being used by a large number of projects and developers worldwide. This huge attack surface might cause security issues to escalate very quickly. As a valuable feature of GHA and CircleCI, R_{18} explained their ability to secure user credentials from being accessible by other developers: "*In CircleCI there is a way you can build the artifacts to a staging area and then you can move the artifacts with the second workflow to where you want to deliver it. That's what we do with GitHub Actions too. We build in one workflow and we have a second workflow which does the upload, shipping or delivery with credentials.*" The availability of a public registry of third-party plugins for the CI tool also introduces an important potential risk [30], since there is little control over the contents of these plugins. For this reason, R_8 valued the way Travis avoids this problem by only providing closed-sourced plugins that are verified by the company itself, therefore reducing the risk of introducing malicious code.

In the following, we present this set of valuable features from the point of view of specific CI tools. Such information will be useful in the context of later question $RQ_{2.1}$ to understand why developers decide to use multiple CI tools simultaneously (e.g., because they have complementary valuable features) and $RQ_{2.2}$ to understand why developers decide to migrate to a different CI tool (e.g., to benefit from valuable features of this CI tool).

GHA. The most recent CI in the list seems to have attracted a lot of attention from respondents for multiple reasons. Since it was developed by GitHub itself, it naturally has very good integration into GitHub. The popularity of the GitHub platform itself among open source developers was reported as a determining factor of choice by 5 respondents. Respondent R_6 decided to select GHA out of familiarity: "*we made the decision at the time that we better move to GitHub instead of Azure DevOps because of the developer familiarity with GitHub over Azure DevOps as a system. So the trade-off there was devel-*

oper familiarity." Moreover, GHA offers free runners, supports a wide range of operating systems (including Linux, Windows, and macOS), and was praised for its ease of use ("*GitHub Actions are so easy to use for CI*" [R_{21}]), its good plugin support through a wide range of actions available on the GitHub MarketPlace, its support for many different languages (including JavaScript, Ruby, and Python), its reliable runners providing fast builds, its support for self-hosted runners, as well as its security mechanisms to avoid exposing user credentials.

Travis. Respondents appreciated Travis' good documentation and the availability of many built-in features. Many respondents appreciated its good integration into GitHub. Indeed, given that its integration with GitHub used to be better than the other available alternatives, Travis used to be the default choice for GitHub repositories at the time. Nowadays, GHA has outperformed Travis in terms of integration with GitHub. Many respondents also praised its good free tier support at the time they were using it (often many years ago). R_8 reported a clear and simple interface, easy configuration, and good default setting for Ruby projects. R_{22} agreed that "*[Travis] was extremely easy to set up with one single configuration file at the root of the project*".

Jenkins. Some of the valuable features of Jenkins that were praised by respondents were its ease of use, its customizability, the availability of many plugins, and it brings a good user experience, even for non-technical users. It also offers self-hosting ability, which is attractive to companies that want to exert full control over the CI automation, notably in order to comply with their service-level agreements related to downtime, service provider response time, security, and turnaround time.

Azure DevOps. This CI tool was mostly reported by respondents working in companies that had a contract with Microsoft for their infrastructure. The valuable features of Azure DevOps were its integration with other Azure tools (4 respondents), good standard built-in features, good support for plug-ins, a better integration with GitHub compared to CircleCI, good runners for Windows and macOS, easy step-by-step configuration, a history of work items and deployments[1], ease of defining multiple build environments, and good separation between CI and CD configuration. With respect to the latter feature, R_{18} specifically appreciated Azure DevOps' "Release pipelines"[2] as a way to facilitate the deployment automation.

GitLab CI/CD. Unsurprisingly, GitLab CI/CD was only mentioned by developers using the GitLab social coding platform. Respondents appreciated its good free runners, a secure debug process, its ease of use in comparison with Jenkins for using private resources, its support for Docker containers. They also

[1] See `https://docs.microsoft.com/en-us/azure/devops/boards/queries/history-and-auditing` and `https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/deployment-history`

[2] `https://docs.microsoft.com/en-us/azure/devops/pipelines/release/?view=azure-devops`

appreciated its self-hosting ability which distinguishes GitLab CI/CD from its competitor GHA.

CircleCI. Respondents specifically appreciated CircleCI's support for Windows, macOS, ARM, and Docker containers. They also valued its user interface with good visualisations, its nice feedback loop with GitHub, its speed, its facility for creating complex parallel and conditional pipelines, and the concept of workspaces.[3]

Drone. The reported valuable features for Drone were its support for ARM architectures, its self-hosting ability and an intuitive interface.

Hudson. R_{22} reported having used Hudson for a long time for closed source projects and private repositories, and appreciated the CI's self-hosting ability. R_{15} appreciated the easy setup and configuration because Hudson was developed in Java and the team had experience working in Java environment.

TeamCity. Only one respondent reported on the valuable features of TeamCity, valuing the user-friendliness of the tool, as well as its self-hosting capabilities.

AppVeyor. This CI tool was reported by four different respondents as the CI tool that historically used to have the best support for Windows. Since no other valuable features were reported for AppVeyor, this seems to be the main reason that caused developers to use it.

Other CI tools. Considering the CI tools being mentioned by single respondents (and hence not shown in Table 4), R_5 valued Concourse because of its good visualisation and ability to set personal triggers for pipeline activation. The ability to have completely independent pipelines in Concourse also enables to connect multiple repositories or Docker images to one pipeline and use user-defined or pre-defined triggers to start the pipeline. Percy was praised by R_8 as the only available CI tool with specific visual CI abilities: "*It basically would render your web page or you define a number of views that you want to test, take a screenshot essentially and then in your branch that you're working on it would do the same and it would compare the screenshots between the branches [...]. If you're assuming when you deploy or when you make a change, if you've broken something, say in your CSS, then you could have some visual bugs that wouldn't show up in the automated testing. So this is a good way of catching some of those more obscure CSS bugs*".

[3] `https://circleci.com/docs/2.0/workspaces`

The CI tools that were most popular among respondents came with the biggest set of valuable features (such as ease of use and support for a wide range of hardware architectures and operating systems). With the exception of Jenkins, the most valued CI tools are cloud-based solutions (GHA, Travis, Azure DevOps and GitLab CI/CD) that come with a good integration with their hosting platform and a good free tier. These solutions also feature a good support for reusable plugins, with the exception of GitLab CI/CD that on the other hand offers a self-hosting ability.

Less popular tools were still considered valuable because they offered specific features that were not available at the time in the other competing CI tools, such as Windows build support, visual testing for UI design, or support for multiple repositories in a single pipeline.

These results differ from earlier studies in that we provide a tool-specific analysis. We also observe a clear shift of the CI landscape towards more cloud-based solutions, with a free tier offer for open source projects, tight integration in the social coding platform, and a registry of reusable components to facilitate creating CI workflows.

$RQ_{1.5}$ What are the reported shortcomings of CI tools?

We asked respondents about the shortcomings they experienced in the CI tools they had used. Table 5 reports on these shortcomings, grouped into various categories that we described hereafter. Some reported shortcomings were considered so severe by the respondents that they caused the project to migrate to a different CI tool. Those migration reasons will be discussed in more detail in $RQ_{2.2}$. It is worth to mention that this list of shortcomings is inevitably incomplete, since respondents may have forgotten to report some shortcomings while focusing on the major ones they had experienced. Moreover, it may be the case that some reported shortcomings are no longer relevant today, given that CI tools continued to evolve and improve.

Hard to configure. Configuration difficulties were reported for several CI tools. The initial build configuration was reported to be difficult to create in TeamCity. GitLab CI/CD did not have a simple workflow. GHA was reported by two respondents to be difficult to configure because too many options are available, and because of the lack of an appropriate default initial configuration. Jenkins was reported by five respondents as difficult to configure.

Too slow. Given that speed is considered as a valuable feature of CI tools (see Table 4), it is not surprising that many respondents mentioned slow runners as a shortcoming of some CI tools. This was the case for Hudson, GitLab CI/CD, Travis and Jenkins. One respondent also mentioned that even GHA was too slow for their specific needs, even though three other respondents explicitly acknowledged the speed of GHA as a valuable feature.

Table 5 Shortcomings of CI tools as reported by respondents. (CI tools that were used by only one respondent are not listed in the table.)

shortcomings	**GHA**	**Travis**	**Jenkins**	**GitLab CI/CD**	**CircleCI**	**Azure DevOps**	**Other**
hard to configure	8 15		6 11 12 17 21	8			1: TeamCity
too slow	10	5 18 19 22	16	22			22: Hudson
unsatisfactory user experience	2 5 8 10 15 18	8	16	2 22		17	
restrictions of free tier	4 21	4 9 10 11			22		
security issues	6 8 15	13					4: Bamboo
lack of scalability		18	1 12	4 13			
plugin problems	8	8	12			15	
no support for specific architectures/OS	4 21	19					
feature stagnation		4 13					9: AppVeyor, Drone 5: Concourse
lack of reliability		4 5 8 9 13 21					
insufficient access to logs	15						13: TeamCity
lack of GitHub integration						9	19: Zuul

Unsatisfactory user experience. Different CI tools were reported to have an unsatisfactory user experience for a wide variety of reasons. Jenkins was reported to have an outdated user interface design. GitLab CI/CD was reported to have a cluttered user interface and no web interface for defining workflows. For Travis, one respondent reported a bad user experience since most configuration tasks for integrating the CI tool into GitHub needed to be done manually. For Azure DevOps, one respondent regretted the absence of YAML-based configurations of workflows. For GHA, two respondents mentioned a too sparse user interface for workflow configuration and four respondents reported no good visualisation of workflows. Additionally, one respondent mentioned the difficulty to start using GHA due to insufficient documentation (especially in the early days of GHA).

Restrictions of free tier. Respondents reported restrictions imposed by the free tiers of CI tools on the build time, the amount of available memory, and the number of runners that could be used in parallel. This was the case for Percy, CircleCI (which did not support macOS under its free trier), GHA and Travis. Travis in particular was agreed upon by many respondents to have imposed many restrictions on its free tier after the company's decision to change its policy towards support for open source projects. The reasons for these imposed restrictions will be discussed in detail in Section 6.3. In a nutshell, Travis replaced its free tier for OSS project builds, that used to offer a fixed number of minutes *per month*, with a higher fixed number of minutes *for life.* At the same time, Travis restricted the set of projects that they qualify as open source, as reported by R_{19}: "*because how they defined open source, they wouldn't even define [OUR PROJECT] as an open source project [...] because according to their requirements, if someone was paid to work on the project like I am, it wouldn't qualify for the open source tier at Travis.*" Given that OSS projects have a very limited budget, R_{19} saw no other choice but to migrate to another CI tool.

Security issues. Several respondents mentioned security concerns related to CI usage. They did so for Travis, GHA and Bamboo, but any other CI tool is likely to suffer from security issues to some extent. Travis was reported by R_{13} to lack correct communication about an important data breach "*they had a security breach [. . .] and they did not communicate properly about this*". GHA was reported by three respondents to have security issues related to working with credentials and self-hosted runners. R_6 explains that "*if somebody already developed a [GHA] Action, you can just plug it into your project and that works great for an open source project because the software is open sourced. You're not worried of the vulnerability and GitHub takes on the responsibility for all the security problems that you would kind of encounter if you tried to run your own CI platform.*" However, "*you can't do that in enterprise. You basically don't trust your software to run on anybody else's machines, or on virtual machines you don't control.*" In addition to this, the reliance on reusable components (e.g., Actions) to automate development activities in software projects increases their attack surface considerably.

Lack of scalability. Scalability refers to the ease of seamlessly and transparently increasing the capacity of CI tools to accommodate for bigger builds, for example by offering longer build times, more parallel runners, and more computing resources (processing power and memory) for the CI process. Scalability issues were reported for three CI tools: Jenkins, GitLab CI/CD and Travis that was reported by one respondent to have a memory bottleneck. Respondent R_6 acknowledged that scalability necessarily comes at a certain cost. Moreover, it is more difficult to achieve in self-hosted solutions, compared to cloud-based CI tools: "*Often you evaluate for capability. Hey, is this system going to be able to do what we need to do? You evaluate for its scalability, meaning yes, it can run one build, but can it run 1000 builds per day? And third is going to be cost, and that comes in two flavors. There's the actual out of pocket operational expenditure of running this system. And then there's the maintenance and continuous support for the system from the developer or maintainer perspective. Usually one of those evaluation criteria is not met by the target system for whatever reason. Most frequently it's a scalability one.*" Respondent R_9 reported another scalability issue related to data bandwidth and the absence of automated caching of compiler outputs: "*If you run CI a lot of times, you download quite a lot of data from the Internet. Because you have packages that you install during your CI run [...] and you basically use up a lot of bandwidth and data. And, again, when you compile software, for example C++ projects, it takes up a lot of time because it's a computationally intensive process.*"

Plugin problems. Problems with plugins were reported for multiple CI tools for different reasons. Jenkins was reported to be too barebone, requiring the user to need many plugins from the start. For Travis, plugins are only updated by the company itself, providing limited freedom to the user. Azure DevOps was reported not to have the ability to write and customize plugins (as is possible in GHA, for example). In case of GHA, R_8 reported not being fond of having community plugins and preferred those CI tools that only offer built-in plugins: "*Travis did provide [. . .] a good amount of things already installed on the CI machine. So I didn't need to use [. . .] all of the different plugins that you can use. [. . .] Most of the GHA plugins are kind of community ran on open source repositories. That makes me very nervous of using them.*"

No support for specific architectures or operating systems. R_{19} regretted Travis' lack of support for the FreeBSD OS and ARM hardware architectures. R_{21} regretted GHA's lack of support for specific OS and hardware architectures, and R_4 regretted the absence of support for HPC binaries. Several respondents agreed that many CI tools have recently become better in supporting the major operating systems (Linux, Windows, macOS). Still, most CI tools remain limited when it comes to less common operating systems (e.g., Solaris and FreeBSD) and hardware architectures (e.g., specific GPU processors). Moroever, as pointed out by R_{19}, some projects require to build and deploy on such a wide diversity of OS that no single CI tool is able to satisfy these needs: "*[name of project] is a portable project. It runs on so many architectures and operating systems that we don't have nearly that coverage in CI services*".

Feature stagnation. Four CI tools (Travis, Drone, AppVeyor, and Concourse) were reported by respondents as suffering from a lack of new features being introduced, causing projects to move away from them. For example, R_5 reported that: "*one of the drawbacks of Concourse is that I don't see a lot of active development anymore on the tool itself. [...] It has been in development for a number of years, but now in the past year, it also became stagnant.*"

Lack of reliability. Travis was the only CI tool reported by many respondents to have become less reliable for a variety of reasons. Two respondents reported a decrease in service quality. Respondent R_8 complained about the company changing its way to support webhooks "*They stopped all of their webhooks that basically just stopped doing CI for almost all of my projects on any builds.*" R_5 reported problems with the CI's customer service, since they took a long time to answer or not even answering about quality of service problems. Two respondents complained about the flakiness of Travis. For example, R_4 stated that he was "*seeing a bunch of reliability problems in Travis where jobs would* *flake out and we would have to rerun them.*" Two other respondents mentioned unreliability of Travis without pinpointing specific reasons.

Insufficient access to logs. For TeamCity, respondent R_{13} regretted not having access to build logs and build results: "*we did not have access to the build logs and the build results. It was quite painful to not have that feedback loop, but still knowing it was running in the background.*" For GHA, respondent R_{15} reported the absence of a deployment history as problematic: "*GHA does not support the history. In Azure DevOps I remembered we had the history. I mean that you had a facility when you wanted to go back in time and just deploy one release that you had.*"[4]

Lack of GitHub integration. R_{19} reported a lack of integration with GitHub for Zuul: "*their integration with GitHub has some kind of flaw. In many cases when we run CI jobs on Zuul they don't show up like the other jobs do on GitHub.*" Azure DevOps was also reported by one of the respondents to lack proper integration with GitHub.

Different CI tools suffer from different shortcomings, such as configuration problems, slowness, unsatisfactory user experience, restrictions on its free tier, security issues, insufficient scalability, plugin problems and many more. Travis was considered to be the most problematic by respondents, mainly suffering from lack of reliability, slowness, restrictions on its free tier, and feature stagnation. GHA was also reported to exhibit several shortcomings, mostly in relation to an unsatisfactory user experience and security issues, as well as some missing desirable features. For Jenkins, the main reported shortcoming was its configuration difficulties.

These findings are in line with those of earlier studies. Without focusing on CI-tool-specific shortcomings, Hilton et al. [37] identified general shortcomings of CI usage, such as configuration problems, slowness, security issues, and lack of good integration. They identified some additional shortcomings that were not reported by our respondents such as the difficulty of troubleshooting CI build failures. On top of this, Elazhary et al. [16] identified some other shortcomings such as lack of features for UI testing, bottlenecks for committing due to PR reviews, and scalability issues due to resource restrictions. One of the shortcomings that we did not observe in earlier studies were the plugin problems mentioned by several respondents. Specifically for Travis, our findings are in line with Widder et al. [27] who reported Travis being slow, having unsatisfactory user experience, not supporting specific architecture or OS, and feature stagnation. Our respondents reported all these shortcomings, as well as several others. Specifically for GHA, Kinsman et al. [12] reported discussions around problems and frustrations with broken builds, errors and other

[4] Azure's Deployment History feature enables to go back in time to allow to redeploy a release that was for example available one year ago, just by selecting that deployment in the history. See `https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/deployment-history?tabs=azure-portal`.

problems. However, they did not discuss these shortcomings in depth, making it difficult to compare them with our own findings.

5 Goal G_2: Why and how are CI tools being co-used and what are the reasons for migrating to other CI tools?

This section tackles research goal G_2, aiming to understand the reasons for using different CI tools together, as well as the reasons and difficulties for migrating to another CI tool. We will do so by providing answers to the following research questions:

$RQ_{2.1}$ Why are multiple CI tools co-used simultaneously?
$RQ_{2.2}$ Why do software projects migrate to a different CI tool?
$RQ_{2.3}$ What are the difficulties in carrying out a CI migration?

These research questions will be addressed in the next three subsections.

$RQ_{2.1}$ Why are multiple CI tools co-used simultaneously?

In their quantitative study of CI usage in 92K GitHub repositories, Golzadeh et al. [11] found that co-using CI tools (i.e., making use of several CI tools at the same time) is common practice. This is surprising since one might intuitively expect all CI tools to provide similar services. We are not aware of any published qualitative analysis aiming to understand the reasons behind such co-usage. We therefore inquired the interview respondents about the reasons behind this phenomenon. 13 out of 22 respondents confirmed that, in at least one of the projects they were involved in, multiple CI tools were being used simultaneously. In this section, we explore all CI co-usages that have been mentioned by the respondents in order to understand the need for such co-usage.

Figure 2 reports on the number of respondents making use of multiple CI tools simultaneously.[5] We observe that the co-usage of CI tools is not restricted to the most popular ones. The combination of (Travis, AppVeyor) was reported 4 times, and the combinations of (CircleCI, AppVeyor), (GHA, CircleCI), and (Azure DevOps, AppVeyor) were reported 3 times. These findings corroborate the ones of Golzadeh et al. [11] that already observed that Travis and AppVeyor was the most frequent case of co-usage, and that Travis, AppVeyor, GHA and CircleCI were involved in most (92.1%) co-usages.

Focusing on the need for co-using CI tools, Table 6 summarises all reported reasons for co-usage of CI tools. We observe that *supporting multiple operating systems* is the most frequently mentioned reason for co-usage (5 respondents). Most of the respondents mentioned the need to build software products at least for Linux, macOS and Windows. Older versions of many CI tools had

[5] Whenever 3+ CI tools are used together, each pair of CI tools is reported individually.

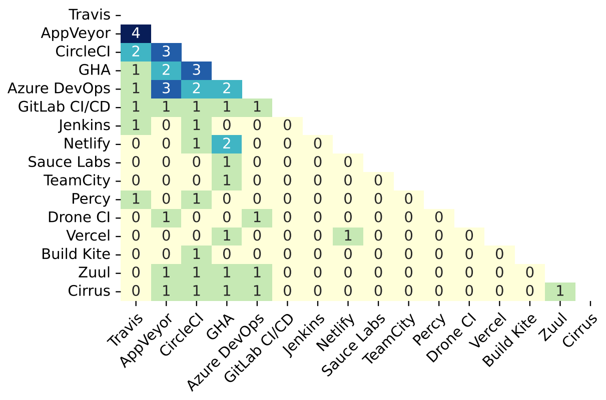

Fig. 2 Number of respondents co-using a pair of CI tools

Table 6 CI tool co-usage reasons reported by respondents.

reason for CI co-usage	respondent IDs
Supporting multiple operating systems	4 9 13 19 22
Complementary functionality	8 10 13
Having a backup CI tool	1 18 19
Countering resource limitations	4 19 21
Supporting specific hardware architectures	4 13
Testing a CI for potential migration	1

limited support for some of these OS. For many years, most CI tools have only supported a single operating system (usually Linux, macOS or Windows), and that is the reason why R_9 reported to use AppVeyor for Windows builds, and Travis for Linux and macOS. Nowadays, most CI tools support the three main operating systems. As an example, GHA initially started with support for Linux only, and added support for macOS and Windows later on. In the case of Travis, support for macOS was added since April 2013[6] and support for Windows in October 2018[7]. For GitLab CI/CD, Windows runners were added in beta version in January 2020[8] and macOS support was added in August 2021.[9]

Additionally, some respondents required support for more specific operating systems. For instance, while R_{13} used CircleCI for Linux builds, this respondent required BuildKite for FreeBSD builds. Two respondents also mentioned the need to support specific hardware architectures (e.g., specific CPU or GPU processors) such as ARM64 and specific AMD or Intel processors. R_9 reported using Drone for ARM 64 CPUs, and lately also Azure DevOps to

[6] `https://saucelabs.com/blog/announcing-travis-ci-for-mac-and-ios-powered-by-sauce-labs`
[7] `https://blog.travis-ci.com/2018-10-11-windows-early-release`
[8] `https://about.gitlab.com/blog/2020/01/21/windows-shared-runner-beta`
[9] `https://about.gitlab.com/blog/2021/08/23/build-cloud-for-macos-beta`

leverage better support for these targeted platforms. R_4, who is involved in offering HPC as a service, has to use many CI tools simultaneously in order to support the specific architectures required by their project: "*We need GPU nodes. We need AMD GPUs which no cloud has. We need Intel GPUs which no cloud has.*"

The fact that CI tools tend to offer *complementary functionalities* is another reason for co-using CI tools. This could be either because the CI tool put in place does not offer some specific features required by the project, or because these features cannot be used in the way the developers expect. As an example, R_8 reported using Percy in complement of Travis and CircleCI because Percy is one of the few CI tools that provides support for *visual testing*, a technique that helps developers to ensure that a graphical user interface appears to the end-user as originally intended. Another example reported by R_{10} and R_{13} concerns Netlify and Vercel, two CI tools that are designed specifically for deployment of web applications. They notably facilitate the dynamic scaling of the application based on the number of connected users, or based on the number of database requests. R_{10} relies on both Netlify and Vercel in complement of GHA: "*If this is some kind of website or web app then we use Vercel or Netlify for the deployment aspect of it.*" R_{13} also co-uses Netlify alongside CircleCI and GHA for a better overall CI experience: "*we are very happy with the resources that we have in CircleCI [...], but we are also happy with the integration that we have in GitHub, the caching that we can have in GitHub Actions and we are also very happy with the specialised nodeJS features that we get at Netlify*".

Another frequently reported reason for co-using CI tools is *having a backup CI tool* in the case the main CI tool being used becomes out of order. R_{19} indicated relying on a custom-built in-house CI tool "*so we still have that too as a sort of additional backup way of testing stuff*". R_{18} reported they kept using Travis alongside CircleCI and Jenkins as a backup for six months until the project team decided that it was no longer needed.

Some respondents mentioned the need for more (free) resources as the reason to co-use CI tools. For example, R_4 "*realised that co-usage can help you to have more runners which lets you increase the amount of jobs you are running*". R_4 uses self-hosted CI tools side-by-side with cloud-based solutions to counter the time limitations imposed by the free tiers of CI tools: "*we attach our own resources and we do it via GitLab CI/CD because the time limit is greater than what GitHub Actions allows*". Similarly, R_{19} reported that "*we added more CI services, so we got more parallelism so that we would complete all jobs sooner. That has been one of the primary reasons why we still use a lot of them because it makes sure that we can run more jobs in parallel until they complete*". R_{21} co-used Sauce Labs in complement to GHA because at that time GHA did not yet provide the ability to use the Windows VM during development for free in open source projects. According to this respondent, "*[our project has to] work in all the browsers and [...] Sauce Labs gives free stuff to open source projects [...] they let me run tests on Windows*"

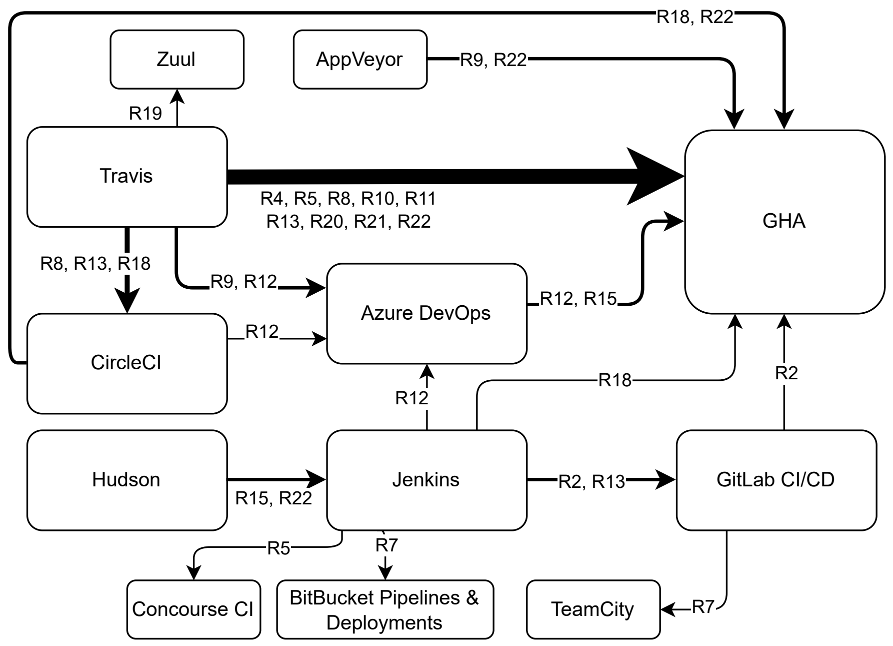

Fig. 3 Reported completed migrations between CI tools.

A last reported reason for co-usage is to *test a CI for potential migration.* R_1 reported introducing TeamCity in complement to GHA since "*it would allow us to migrate if necessary*".

It is quite common practice to use several CI tools in parallel. Most combinations involve Travis, AppVeyor, GHA or CircleCI. The most frequent reasons are to cover more operating systems, to access complementary features, and to benefit from more computing resources.

$RQ_{2.2}$ Why do software projects migrate to a different CI tool?

We observed in $RQ_{1.1}$ that most respondents have used several CI tools through time. We asked them explicitly whether they co-used these different CI tools (see $RQ_{2.1}$), and whether they migrated from one tool to another. Since developers may decide to migrate to another CI tool for different reasons, we also asked the respondents to share their experience on why they migrated.

Overall, respondents reported a total of 32 completed migrations in different projects (some respondents reported multiple migrations), involving 12 different CI tools. Figure 3 shows the migration paths. We observe that the reported migrations originate from 7 distinct CI tools and lead to 9 distinct CI tools. Most migrations originate from Travis (15 out of 32) and lead to GHA

(17 out of 32). The most frequently observed migration pattern is from Travis to GHA (9 migrations), corroborating the findings of Golzadeh et al. [11].

Overall, we observe a general tendency to move from self-hosted CI tools to cloud-based solutions since most of the completed migrations are to cloud-based CIs (e.g. GHA) and at least half of migrations from Jenkins (a well known self-hosted CI tool) are towards cloud-based CI tools (namely GHA, Azure DevOps, and Bitbucket Pipelines). Free tier cloud services have the advantage of not needing to configure and maintain a local CI server, which can be quite costly for small teams and projects in terms of personnel and hardware resources. R_9 therefore prefers using the free tier of a cloud-based CI service, rather than spending any budget on that resource: "*since everything we're doing is open source and most of these CI providers have an offer for open source projects to provide them with free hardware or CPU time we usually don't spend any money on hardware*". Moreover, cloud-based CI services are usually more scalable and adaptable to the actual resource needs of the project, as mentioned by R_5: "*just to make sure, we run on cloud infrastructure, so if we need to scale, we scale.*"

We asked respondents to share the reasons why they migrated. Table 7 shows the 10 reported reasons to migrate, as well as the source and target of each migration. Some migrations are reported more than once since there were multiple reasons that drove the decision to migrate.

The most frequent reason that was reported to migrate is to *have less restrictions on free tier*, mentioned by 8 different respondents for 10 different migration cases. All these cases correspond to migrations away from Travis. As discussed in $RQ_{1.5}$, the change in Travis' free tier imposes so many restrictions on open source projects that it leads them to migrate to another CI. For example R_{19} mentioned "*I figured the project could use sponsored money or donations to pay for it, but I felt it would be more responsible for our project to not spend that money on Travis, but rather to save the money and just move to another free service instead.*" Half of the migrations away from Travis lead to GHA (5 out of 10 cases), the remaining ones being to Azure DevOps, CircleCI and Zuul. Another reason to migrate away from Travis is to "*use a more reliable CI tool*", reported by 7 respondents. This is a consequence of the many reliability issues identified in $RQ_{1.5}$ for Travis. This had led respondents to migrate away to more reliable CI tools such as GHA (5 reported cases) and CircleCI (2 reported cases).

The second most frequent reason to migrate to another CI tool is to obtain a "*better integration with hosting platform*" (8 cases). This reason refers to the integration of CI tools within the social coding platform used by the respondent, typically GitHub, GitLab, Azure or BitBucket. The target of these reported migrations is almost exclusively GHA, likely due to its deep integration within GitHub, the most popular hosting platform.

The need for "*better support of multiple platforms*" also explains several migrations. For 5 out of 6 cases, GHA was the target of choice, as it supports the most common operating systems such as Linux, Windows, and macOS. One respondent (R_{12}) actually migrated from CircleCI to Azure DevOps with

Table 7 Reasons for completed migrations.

migration reason	migration from	migration to	respondents
Having less restrictions on the free tier	Travis	GHA	10 11 13 21 22
	Travis	Azure DevOps	12 19
	Travis	CircleCI	8 13
	Travis	Zuul	19
Using a more reliable CI tool	Travis	GHA	4 8 13 20 21
	Travis	CircleCI	13 18
Better integration with hosting platform	AppVeyor	GHA	9 22
	CircleCI	GHA	18 22
	CircleCI	Azure DevOps	12
	Azure DevOps	GHA	12
	Jenkins	GHA	18
	Travis	GHA	22
Better support of multiple platforms	AppVeyor	GHA	9 22
	CircleCI	GHA	22
	Jenkins	GHA	18
	Azure DevOps	GHA	12
	CircleCI	Azure DevOps	12
Decreasing the amount of CI tool co-usage	CircleCI + Jenkins	GHA	18
	Travis + AppVeyor	Azure DevOps	9
Having better features	Azure DevOps	GHA	15
	Jenkins	Concourse	5
Moving to a successor CI	Hudson	Jenkins	15 22
Making the project open source	GitLab CI/CD	GHA	2
Moving to a new ecosystem	GitLab CI/CD	TeamCity	7
Reducing the maintenance burden	Jenkins	Azure DevOps	12

the aim of a better integration with GitHub and a better support for Windows, but this was before GHA existed. They migrated from Azure DevOps to GHA a bit later.

Another reported reason to carry out migrations relates to the CI co-usage that we analysed as part of $RQ_{2.1}$, namely to "*decrease the amount of CI tool co-usage*". While the results of $RQ_{2.1}$ highlighted the need to co-use multiple CI tools for specific reasons, it comes at a certain cost and increased effort: "*Co-usage introduces at least two difficulties. You need to maintain both, they have sometimes different syntax in the YAML files, so you have to have more knowledge so it's more work, that's sort of the first issue. The second issue I see is for example for code coverage. If you do code coverage on both platforms and you want to merge your code coverage, it might be difficult, [...] whereas when you have only one CI/CD provider, it's much easier because there is only one workflow*" [R_{18}]. Therefore, project maintainers keep track of the evolving functionalities of the available CI tools in order to seize the opportunity to reduce the maintenance overhead caused by using multiple CI tools. For example, R_{18} replaced a combination of CircleCI and Jenkins by GHA, while R_9 replaced Travis and AppVeyor by Azure DevOps since they wanted "*to unify our pipelines and have everything in one place*". In both cases, the new

CI tool supported all the needs that were previously covered by the two CI tools being co-used.

"*Having better features*" was reported as a migration reason twice. R_5 reported moving from Jenkins to Concourse because "*I really want a good visualization from the whole flow. Concourse gives me that, and not only per single repository. That's the critique I have to most tools. Pipelines are linked to single git repos. Concourse not. Pipelines are completely independent from your all of your code basis*". R_{15} reported a migration from Azure DevOps to GHA because " *[they] knew some news that Microsoft was going to invest on GitHub Actions and not investing lots of effort on Azure DevOps. The features that GitHub Actions provides for writing and customizing plugins [...] encouraged our team to decide to have a migration.*"

Other noteworthy reported migration reasons were:

– *"To move to the successor CI tool".* This reason was mentioned by two respondents that migrated from Hudson to its next generation open source successor Jenkins.
– *"Making the project open source*" was the reason that caused R_2 to migrate from GitLab CI/CD to GHA: "*we wanted to have something that we can show the open source code for the DevOPS pipeline. Since GitHub provides a free runner, and the open source code of the application is on GitHub, we went there.*"
– R_7 reported "*moving to new ecosystem*" as the reason to migrate from GitLab CI/CD to TeamCity since they wanted to make use of the full JetBrains tool suite.
– R_{12} reported *"reducing the maintenance burden"* as the main reason to migrate from Jenkins to Azure DevOps: "*So what drove a migration from Jenkins to Azure DevOps was the maintenance burden of Jenkins. I think we almost had one person full time, just maintaining Jenkins.*"

Respondents are constantly looking for more appropriate CI tools. Most of the reported migrations are away from Travis and towards GHA, a consequence of Travis' change in free tier and reliability issues. Migrations towards GHA are primarily due to its generous free tier, its deep integration with GitHub, and its support for the most common operating systems.

$RQ_{2.3}$ What are the difficulties in carrying out a CI migration?

In $RQ_{2.2}$ we observed that many respondents migrated from one CI tool to another one. Because the different CI tools have different philosophies, approaches and configuration files, it might be difficult to migrate to another CI tool. We therefore explicitly asked the respondents to report on their experience.

Many respondents (R_8, R_{10}, R_{13}, R_{21}) reported having faced no real problems in moving from one CI tool to another one. For instance, R_2 reported

"*I think it was around 3 days. And the reason it was short is because the DevOps pipeline was very simple so we just installed*". Similarly, R_5 described their migration was not a hard process. Given that the destination CI tool was already being used by their company, so they had a basis on which to build specific pipelines for their project: "*we have to create pipelines for ourselves for our code [...] but there is already quite some investment in a number of standardized pipelines, it's not really that we have to start from zero. We can start by duplicating a pipeline and adapting here and there for some tooling that we are running.*" [R_5].

The remaining 18 respondents did mention having faced difficulties during CI migration, but the reported reasons were very diverse. One recurrent reason had to do with the *learning curve* to master the syntax of the new CI tool. Many contemporary CI tools use a YAML-based syntax to describe their workflows or pipelines (e.g., GHA and Travis), while others may use a totally different way to specify CI pipelines, and the differences in syntax and semantics were reported to cause migration difficulties by several respondents:

R_{13}: "*There is no standard way to publish libraries because you want to still reuse pipelines between jobs, between software. In CircleCI it's called orbs, in GitHub Actions this is the Action libraries, in Jenkins it's another one, so there is always a learning curve, even if you know the command to type in the CI. You need to learn the CI tool [...] that really takes a lot of time*".

R_9: "*Sometimes for example environment variables are differently set in the CI systems or some other minor differences between the providers.*"

R_{18}: "*Something interesting you might want to look at are the commits of somebody doing a migration. You will see that you do a lot of typos and try to run the CI/CD 20 times until it works once. You copy-paste some examples from the Internet, you adapt it, but you forget to like there's a lot of details. It's often YAML files that are really prone to mistakes. So you make [lots of] commits until you get to the result you want to have. And there's no way to pre-test it on your local machine. So you just commit, push, wait for the build to run, and then look at the results. So that's why a migration might take some time*".

Solutions are being proposed to reduce this learning curve. For instance, dagger.io provides a way to unify workflow specifications across different CI tools, by offering cross-language instrumentation through dedicated APIs. For example, developers may use the Go SDK to develop all of their CI/CD pipelines using the Go programming language, or the CUE SDK to use the CUE configuration language. In both cases, it avoids needing to know and learn the specific YAML (or other) variants being used by CI tools.

R_{10} reported on the lack of an easy way to ensure the correct execution and behaviour of pipelines within the CI migration target: "*Difficulties when you migrate a CI/CD tool is just the time it takes to verify that it works. Because usually [...] you have to do changes to your repository, and then you have to wait until the CI/CD tools pick up the changes, run the script and tell you back. So the feedback cycle is just slow. Or when I move the automated* *releases script, semantic release, from Travis to GitHub Actions, I actually *had to* do a release to test, to verify that it works. So that took time.*"

R_{12} experienced migration difficulties due to *completely different architecture and security models*: "*The worst migration that I've done is from Travis to Azure DevOps. That was so difficult because they are completely different systems with different security models and different architectures and that took a whole team working for a week to migrate.*"

Three respondents actually considered migrating to a new CI but, in the end, decided not to for various reasons. R_6 considered Azure DevOps as a replacement of a proprietary in-house CI tool used in a big commercial company. The "*lack of developer familiarity with the new CI*" and the "*lack of scalability of Azure DevOps*" were the main reasons holding the company back from migrating. Azure DevOps' capabilities did not match the company's needs for handling a high number of builds per day and having enough flexibility and scalability. R_{19} explained that their company was using Azure DevOps, GHA and a custom-built CI tool at the same time. They were considering a definitive migration towards GHA to reduce the number of co-used CI tools but eventually decided against the migration, in order to keep the benefit of running multiple builds in different CI tools in parallel. Similarly, R_{18} decided not to perform a migration from Azure DevOps to GHA because the latter missed an important feature for continuous delivery and deployment: "*we pay the Microsoft service [for Azure DevOps] because it's a company [and] so you get the support and everything. If we would go to GitHub Actions, we would switch to the professional paid plan [...] but there is something in Azure DevOps which GitHub Actions does not have. It's the Release Pipelines, which is much more evolved*".

Respondents reported a wide range of difficulties during migration to a new CI tool: the learning curve, fundamental differences between the source and target of the migration, the trial-and-error nature of configuring a new CI tool, the lack of familiarity with the new CI tool, and important missing features for continuous delivery and deployment.

6 Discussion

This section discusses important additional insights about CI usage that we have been able to gain from the interviews. Some of these insights did not align with specific interview questions, and others emerged as side-remarks that we consider nevertheless important to discuss here, since they provide additional insights into why developers use specific CI tools or why they decide to migrate to other CI tools.

Section 6.1 starts by discussing the many aspects surrounding GHA that have caused it to become one of the dominant CI tools today. Section 6.2 explains how the open source nature of projects or the CI tools used by them can play an important role in the CI tool being used. Section 6.3 discusses why some CI tools have been subject to restrictions on their free tier. Section 6.4 presents some potential future directions for CI tools, as suggested by interview respondents. Finally, Section 6.5 reflects on the need to have a sufficiently diverse CI landscape in order to satisfy the varying needs of CI tool users, as well as to reduce the risk of certain CI tools taking a monopoly position.

6.1 On the use of GitHub Actions

Research goal G_2 aimed to understand how and why developers have migrated to different CI tools and, in particular, why GitHub Actions has become the dominant CI tool in the current CI landscape. The CI tool usage reported in Table 4 and the migration cases reported in Figure 3 signal the increasing popularity of GHA. This corroborates the quantitative study by Golzadeh et al. [11] who observed that only 18 months after its introduction, GHA has become the dominant CI on GitHub.

Anticipating the popularity of GHA among respondents, the interview questionnaire included a question about valuable features of GHA that were appreciated by respondents, and that caused some of them to migrate to GHA as their CI of choice. Respondents mentioned a variety of reasons for doing this migration: the excellent integration of GHA into GitHub; the fact that it provides a generous free tier for open source projects; its support of a wide range of operating systems and hardware architectures; the availability of a large marketplace of reusable actions; and the availability of better features than some competing CI tools. Each of these valuable features have contributed to GHA's popularity. Another driver for this popularity was the increasing dissatisfaction with Travis (as reported in $RQ_{1.5}$).

Among many other reasons, the company behind Travis failed to correctly communicate about important security issues. Some of those issues can be very dangerous and impactful, such as the exposure of customer-specific secret environment data such as signing keys, access credentials, and API tokens for a duration of 8 days in 2021.[10] Using GHA instead of Travis is of course no guarantee that security concerns will not arise. For instance, in $RQ_{1.5}$ we reported some potential problems and examples of important security issues related to GHA as well.

We also conjectured that the acquisition of GitHub by Microsoft in June 2018 may have played are role in GHA popularity. Given that GitHub is the most popular hosting platform for OSS projects, its acquisition by a big tech company is likely to have at least some impact on the use of its integrated CI service GHA that was publicly released in November 2019. We asked the interview respondents whether the acquisition by Microsoft was perceived as positive or negative. While 14 out of 22 respondents answered that GitHub's acquisition did not have any impact on their CI choice, 8 respondents did say

[10] `https://nvd.nist.gov/vuln/detail/CVE-2021-41077`

that it somehow played a role, raising both arguments in favour or against this acquisition. On the negative side, some respondents raised concerns against the acquisition. For instance, R_{15} prefers using GitLab for his personal projects because of that: "*Personally, I'm using GitLab for my own project. I don't like the Microsoft concept and owning the company or something like that.*" On the positive side, Microsoft's attitude toward OSS has improved recently. R_{22} reported that "*These last years, Microsoft is really doing huge changes internally to make their reputation change about open source. I think Microsoft is changing its point of view on open source and I think it's for the greater good of open-source developers*". In a similar vein, R_5 reported "*For me there are two versions of Microsoft, before and after Satya Nadella became the CEO. With Satya Nadella as CEO that's for me, the V2 of Microsoft as it became much more open source friendly. [...] So when I heard the news that Microsoft bought GitHub I wasn't really too afraid. I would have been more concerned of that acquisition if it would still have been the V1 Microsoft with Steve Ballmer and all that.*" Another positive aspect of the acquisition is that it has enabled GHA to grow rapidly in functionality and performance. R_{11} appreciated the fact that "*every month GitHub is releasing something new: the code Explorer, GitHub Actions, [. . .] So at least the switch to Microsoft was good to get a bit more into the business.*" R_{13} confirms this: "*Now we actually have GitHub Actions. We have a lot more performance things in GitHub that we had in the past, so for me that change was kind of fine.*" This vision is shared by R_9: "*I think the impact of Microsoft buying GitHub so far has been pretty positive, and it has only made GitHub more useful.*"

Next to GHA, Microsoft is offering Azure DevOps as a competing CI product that is part of the Azure cloud-based ecosystem, and many respondents reported having used it. One could wonder how sustainable it is for a company to continue supporting two competing CI solutions with similar functionalities. We observed two cases of respondents having migrated from Azure DevOps to GHA (see Fig. 3). Respondents that used Azure DevOps valued its tight integration in the Azure ecosystem and the technical support offered by Microsoft. For example, R_{14} reported: "*In my current company we are using whatever tool Microsoft is providing. One reason is that we are using Azure Portal, Azure DevOps, Azure anything. So we consider that it's better to build our pipeline using Azure [...] In Azure, there are more plugins and more options for using it in the Microsoft world. Because we are using Azure DevOps, all the repositories are in the same place as pipelines and also all the Scrum boards are in there*".

6.2 Open source nature of CI tools

The open source nature of the software project and/or the CI tool was reported by multiple respondents as playing an important role in the choice of CI tool. Some OSS projects specifically select or impose the use of open source CI tools, as it matches the nature and mindset of their open source policy. For instance,

R_3 mentioned that "*for some practical reason, but also maybe ideological, we like to use open source solutions and have to control on our software that we use.*" Similarly, respondent R_{10} argued "*Philosophically, we don't like the company that bought Travis. They just have different values then we have in our open source projects.*"

OSS projects may also select specific CI tools for more pragmatic reasons. For example, R_2 chose GitHub as a platform to demonstrate their open source code, not because GitHub itself is open source (it is not), but because it is the most popular platform for hosting OSS projects: "*The reason is that we wanted to have something that we can demonstrate, like show the open source code for the DevOps pipeline. And since GitHub provides a free runner, and the open source code of the application is on GitHub we went there.*"

R_5 also argued that the choice of a CI tool depends on whether the project using it is open source or commercial: "*The requirements for an open source project are usually quite different from a commercial project. [... If it is] only a CI for an open source project and you need to build a package that you want to have published in registries, then for example GitHub CI would be completely satisfying. Because GitHub Actions gives me all the building blocks which I need at that moment. But if I want to build a product with all the testing, all the quality gates validation in a number of environments before going to have a real live environment then I would still search for something which gives me the full flow and have more ability to model that full flow.*"

The interviews revealed that 8 respondents involved in OSS projects preferred using the free tier solution of a commercial CI tool (mostly GHA, Travis and GitLab CI/CD) since OSS projects often have very limited financial resources to develop and maintain their software. R_{19} explained the decision-making process for choosing another CI tool due to restrictions imposed on the free tier of Travis: "*I had the choice to either pay for it or not remain on Travis. And then I figured out I had a lot of other options. We could use sponsored money or donations to pay for it, but I felt it would be more responsible for our project to not spend that money on Travis, but rather save the money and just move over to another free service instead*". R_{11} also "*decided to change to go to GitHub Actions that was free*". This shows that changes in the pricing policy or restrictions imposed on the free tier of the CI tool may incite or even force OSS projects to migrate to other CI tools.

On the other hand, commercial projects tend to prefer using paid, commercial CI tools because they offer a better service-level agreement and technical support in case of reliability problems. R_{14} who used Azure DevOps in a company said "*The problem with Microsoft Azure is money. You need to spend a lot of money, but the tools that you are getting from Microsoft [...] are more powerful than the others. [...] If I had money, I'd migrate [my projects] to Azure.*"

Some respondents were not satisfied with any of the existing open source or commercial CI tools. As a consequence, they rely on custom-built CI tools in their companies. These CIs were created to support the specific needs of the company. For example, R_6 mentions that their company "*built its own* *proprietary CI/CD pipeline which now operates to deploy thousands of deployments per day. It was loosely based on Jenkins for a while.*" More specifically, "*it started as a kind of deployment of Jenkins that evolved over time into an actually proprietary system that's built from scratch*".

One aspect in favour of open source CI tools would be the ability to contribute changes back to the OSS project. However, R_6 argued that it not always easy to do so, due to the community's latency of accepting changes: "*Contributing back to the community is not always possible because volunteer-based projects face the challenge that you have of contributing back to popular open source projects. Not everybody's contributions are going to be able to be accepted. So I think a latency is introduced by this.*"

6.3 Restrictions on the free tier of CI tools

Among the main shortcomings that were identified in $RQ_{1.5}$, the restrictions imposed on the free tier of CI tools were frequently reported by the respondents. These restrictions cause many projects to select an alternative CI tool offering more computation resources or more build time. Travis was frequently reported by the respondents as being too restrictive on its free tier. Its decision to impose more restrictions on its free tier in 2020 was even one of the main reported reasons for OSS projects to migrate to another CI. We investigated the rationale behind imposing such restrictions, and we found that the decision was mostly driven by abusive cryptominers.

Li et al. [42] studied the phenomenon of *CI-jacking* which, in other words, correspond to the abuse of CI tool resources for mining cryptocurrencies. Through an empirical analysis of GitHub repositories and log files on CI platforms, they found 1,974 instances of CI-jacking, with an estimated revenue of over $\$20,000$ per month using the computational resources of the CI tools' free tier. This abuse has led CI providers to impose stronger limitations on their free tiers.[11]

For example, Travis motivated its decision to change its pricing model as follows [43]: "*[...] we have encountered significant abuse [...] (increased activity of cryptocurrency miners, TOR nodes operators etc.). Abusers have been tying up our build queues and causing performance reductions for everyone.*" Similarly, in February 2021, the Director of Product Management of GitHub publicly announced strong restrictions on the free tier of Azure DevOps due to "*a high percentage of new public projects in Azure DevOps being used for crypto mining and other activities we classify as abusive*" [44].

Two interview respondents confirmed this abuse by cryptominers, and the harm it is causing to OSS projects whose functioning often depends on the ability to benefit from the resources offered by the free tiers of cloud-based CI tools: "*in recent years people have been abusing a CI/CD solution for mining bitcoins. This is annoying for the open source community because we rely on*

[11] `https://webapp.io/blog/crypto-miners-are-killing-free-ci/`

those tools. Those tools are really critical for the open source communities to continue to build and secure the toolchain" [R_{13}]. In addition to this, R_{21} explained that cryptominers not only affect computational resources, but also impact human resources that need to check for the presence of cryptominers: "*a human has to review the code [of new contributions] and make sure that someone is not trying to install a cryptocurrency miner on our Jenkins installation.*"

6.4 Future of CI tools

Since their inception, CI tools have come a long way, continuously adding new automation facilities to support an increasing range of software development activities. It is beyond doubt that CI tools are widely adopted and play an important role in both OSS and commercial software development [10, 11].

As part of the open-ended closing question of each interview, respondents were asked to share important remarks related to CI tools. Some respondents used this opportunity to share their opinion on the expected future of CI tools and how these tools will become integrated with other software development components.

R_{10} expressed the idea of having a whole *physically independent* infrastructure that can use the full existing features of a social coding platform, including a software development environment, version control system, issue tracking system, online coding environment, and a GHA-like CI: "*[companies and developers] want someone like GitHub to host a version of GitHub for them. That means that you would get your own thing hosted on GitHub's own hardware, but it will be physically separated but still integrated with the github.com platform. The main benefit of that is that you don't need to think about your custom actions runner and things like Codespaces.*[12] *Development in the cloud will become very relevant in future. And when you want to keep self-hosting, it's not only about hosting the git platform. It will be more and more about also hosting all these other things like the CI/CD environment, which is GitHub Actions for GitHub, and the cloud development platform, which is Codespaces. It's going to get harder and harder to self-host [while] a service hosted version will become much more attractive.*" Other companies, such as gitpod.io have also started providing similar cloud development environments, that can be integrated with one's preferred social coding platform (e.g., GitHub, GitLab or BitBucket).

R_6 suggests considering CI in the full software development lifecycle: "*CI/CD is part of a broader system of continuous delivery of software. Looking at it in isolation is like looking at just a portion of a full pipeline. Delivering software begins at this ideation phase and ends when a user interacts with it. And CI/CD has this role to play, but it is not the entire spectrum. So CI/CD needs to be looked at in the context of the rest of the engineering system being used*

[12] `https://visualstudio.microsoft.com/services/github-codespaces/`

to deliver software in a particular place, whether it be OSS or in different proprietary setups." R_5 shared this point of view: "*If I want to build a product with all the testing, all the quality gates validation in a number of environments before going to have a real live environment, then I would still search for something which gives me the full flow [...]*".

R_9 also considered there is room for improvement, notably to address the amount of data that needs to be downloaded each time a CI process is executed: "*there would be automated ways to reduce the amount of data that's downloaded from the providers. Some sort of automatic caching. And the other thing is to also have some automatic caching of compiler outputs. But both things are probably not so easy to do in a generic fashion*". Indeed, CI tools are known to incur a high cost, because of the computational resources they require, combined with the frequency of running builds. Minimizing execution time of CI workflows is crucial, as it enables timely feedback to developers, avoiding them to switch to other tasks while waiting for the CI workflows to complete, which is known to be a costly operation for knowledge workers.

Existing CI tools could benefit from integrating recent research advances that have been made along these lines. For example, CloudBuild, a proprietary CI tool with caching capabilities, was proposed by Microsoft to speed up building and testing software products [45]. Similarly, Gallaba et al. [46] proposed an approach to accelerate CI builds by caching the build environment and skipping unaffected build steps. Abdalkareem et al. [47] proposed another way to speed up build time by skipping commits that should not trigger builds. For this purpose, they developed a machine learning technique that automatically identifies which commits can be safely skipped [48]. In the same vein, Jin et al. [49] introduced SmartBuildSkip, an approach to reduce CI cost by running fewer builds while running as many failing builds as early as possible. Based on an empirical comparison of 10 CI-improving techniques [50] they proposed PreciseBuildSkip [51] as an improvement over SmartBuildSkip.

6.5 On the diversity of the CI landscape

$RQ_{1.1}$ revealed a wide diversity of CI solutions having been used by respondents. $RQ_{1.4}$ further revealed that, even if the most popular CI solutions covered most of the desirable features, there were still valid reasons for using less popular CI tools because they were offering specific valuable features that could not be found in the more popular CI tools. This was confirmed in $RQ_{2.2}$ where respondents identified many reasons for co-using CI tools, such as the need to support specific hardware platforms or operating systems, access to specific features, and the ability to use more computing resources by running multiple CI tools in parallel. This shows the importance of maintaining a wide diversity of CI tools, each having their own set of features, advantages and shortcomings. R_5 even goes one step further by claiming that there still is plenty of room for new contenders in the CI landscape: "*a lot of people think that tools regarding CI/CD is already a well-equipped market. But personally I* *think there is still a lot of room for improvement. If there would be a contender really thinking out of the box [...] I think he would still make a fair chance of getting a decent market share*".

Nevertheless, in response to $RQ_{2.3}$ as well as in Table 4 we observed a general tendency to migrate towards the more popular cloud-based CI tools (such as GHA, Azure DevOps and GitLab CI/CD). The ever stronger integration of these CI tools in their social coding platform, compounded by the fact that workflows and pipelines are increasingly relying on reusable building blocks (such as Actions, Orbs and plugins) makes it more difficult to migrate away from them. This leads to an increased risk of vendor lock-in, that may lead to a monopoly position of some CI providers, ultimately resulting in a lack of innovation due to absence of competition.

GitHub, the *de facto* solution for open source projects nowadays, is a good example of potential vendor lock-in by a private company owning the dominant platform for distributing open source software. R_9 was concerned about this risk: "*The only kind of concerns, obviously, to have the vendor lock-in and kind of monopoly situation.*" R_{11} shared this viewpoint: "*From an ethical point of view it's a pity that GitHub and Microsoft joined together.*" At some point in the future, Microsoft might change its strategy to try to profit from its monopoly: "*There are intangible benefits that Microsoft gets and we'll see if changes happen in the next few years, to where GitHub makes changes to be more profitable and that don't necessarily serve the free software folks. [...] I have mixed feelings about it, on the one hand, it really is convenient having everything integrated at one place. On the other hand, how much do we really want to invest all of open source in a single company?*" [R_{21}]

Nevertheless, it seems like respondents are aware of this risk and still willing to use GitHub, while keeping their options open to move to other CI alternatives: "*so far I think, for us, it's been a positive experience. But we are aware of these dangers and we would be ready to move to another platform if we have to*" [R_9]. Similarly, R_1 reported to "*have alternatives in case there is something that changes in the GitHub CI user conditions. For instance, If GitHub Actions becomes irrelevant or not practical given the conditions of the project, then we know that there is a simple way to just use TeamCity instead of GitHub Actions.*"

7 Threats to validity

Here we discuss the threats to the validity of our work.

Internal validity relates to whether an experimental treatment or condition makes a difference or not [52]. Given that our analysis is based on subjective interviews, the main threat pertains to which questions we asked, how, and in which order. A different set of questions, different order, or even different phrasings could have led to different responses. We reduced this threat by carefully verifying the interview protocol, and carrying out dry-runs on three different persons, before actually starting the study. Moreover, the open-ended nature of the interviews provided ample opportunities for respondents to provide additional contextual information that was not necessarily directly related to the questions being asked.

External validity is concerned with the generalisability of the approach and the representativeness of the results [52]. We strove to have a good balance in respondent profiles covering both open source developers and industrial practitioners. While we strove to have a balanced selection of respondents in terms of geographical distribution, we acknowledge that Eastern Europe as well as the Australasian and African regions are underrepresented in our population. The inclusion criteria that have been used for selecting interview candidates also introduced a deliberate selection bias towards developers with proven practical experience with CI. As a result we cannot claim that the results generalise to less experienced developers.

Construct validity concerns the relation between the theory behind the experiment and the observed findings [53]. Given the qualitative nature of our study, the mean threat pertains to how we have interpreted the responses obtained from the interviews. To mitigate this risk, we have relied on the well-established process of deductive and inductive coding, which involved multiple authors in order to further lower the risk of making incorrect interpretations.

Conclusion validity deals with the degree to which reasonable conclusions have been reached from the collected data [54]. One might argue that having more respondents could have increased the support of our findings. Since we continued interviewing respondents until we reached saturation in the responses, we believe that the conclusions made are reasonable, especially given the qualitative nature of this paper, as we do not aim to show any statistical significance of our observations. Moreover, some of the qualitative findings have been triangulated with quantitative results reported in earlier work.
Geopolitical reasons may have implicitly affected some of the received responses. For example, European respondents are subject to other privacy regulations (GDPR) than non-European ones, which could have influenced their preference toward CI tools maintained in Europe. As another example, the acquisition of GitHub by Microsoft, an American company, could have influenced some respondents in favour or against the use of GHA as a CI tool. Also, two respondents reported not having been able to use some popular commercial CI tools since a ban was imposed on certain countries (such as Iran). This is not a problem per se, since the conclusions drawn from the interviews are actually supposed to reflect and capture this diversity in decisions.

8 Conclusion

This article presented the results of a qualitative analysis aiming to understand the reasons behind CI tool usage, co-usage and migration. The analysis is based on online interviews with 22 experienced software practitioners with proven expertise in CI tools. The interview respondents were involved in OSS as well as in commercial projects, and reported on the use of 31 different CI tools, of which 14 were used by at least two respondents.

The large number of CI tools used by the respondents, the reasons to use them, and the wide range of activities to automate highlight how diverse the landscape of CI solutions is. The main reasons for CI usage were to increase reliability, productivity, security, and speed while reducing cost and effort. The main supported activities were build automation, unit testing, security and quality analysis, dependency management, release management and automated deployment. While the valuable features and shortcomings of CI tools have not fundamentally changed in comparison to previous studies, we observed a clear technological shift towards more cloud-based solutions integrated in social coding platforms (such as GHA, GitLab CI/CD, Azure DevOps).

We observed that it is common practice to use multiple CI tools in parallel in order to support a wider range of hardware architectures or operating systems, as well as to benefit from complementary features offered by the different CI tools and to counter resource limitations. We also observed a migration away from Travis, due to lack of reliability, feature stagnation and restrictions imposed on its free tier. At the same time we observed a migration towards GHA due to its deep integration into the popular GitHub social coding platform, its generous free tier, its build support for the major operating systems, and its support for reusable Actions. The main reported migration difficulties had to do with the learning curve because of the differences between the source and target CI tools.

Our analysis provided qualitative insights into the reasons behind the important changes in the CI landscape that were quantitatively reported in [11]. This changing landscape has opened up a wide range of research opportunities, such as more empirical research on the impact of reusable workflow components (such as GitHub's Actions and CircleCI's orbs), an in-depth analysis of the risks of vendor lock-in, and technical solutions to further speed-up CI execution.

Acknowledgments

This work is supported by the ARC-21/25 UMONS3 Action de Recherche Concertée financée par le Ministère de la Communauté française – Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique, as well as by the Fonds de la Recherche Scientifique - FNRS under grant numbers O.0157.18F-RG43 and T.0149.22 and F.4515.23.

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability Statements

All data generated or analysed during this study are included in this published article (and its supplementary information files). Except for two interview transcripts, the extracted information is available in the article content but the interview transcripts themselves are not accessible to the public due to the interviewees' request.

References

1. Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional, 2007.
2. A Shahin, M.A. Babar, and L. Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017.
3. Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.
4. Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in GitHub. In *Joint Meeting on Foundations of Software Engineering (FSE)*, pages 805–816, 2015.
5. Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *International Conference on Automated Software Engineering (ASE)*, pages 426–437. IEEE, 2016.
6. Moritz Beller, Georgios Gousios, and Andy Zaidman. Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In *International Conference on Mining Software Repositories (MSR)*, pages 356–367, 2017.
7. David Widder, Bogdan Vasilescu, Michael Hilton, and Christian Kästner. I'm leaving you, Travis: a continuous integration breakup story. In *International Conference on Mining Software Repositories (MSR)*, pages 165–169. IEEE, 2018.
8. Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. Continuous deployment at Facebook and OANDA. In *International Conference on Software Engineering (ICSE)*, pages 21–30. IEEE, 2016.
9. Helena Holmstrom, Eoin Ó Conchúir, J Agerfalk, and Brian Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *International Conference on Global Software Engineering (ICGSE)*, pages 3–11. IEEE, 2006.
10. Eliezio Soares, Gustavo Sizilio, Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, 27(3):1–61, 2022.
11. Mehdi Golzadeh, Alexandre Decan, and Tom Mens. On the rise and fall of CI services in GitHub. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022.
12. Timothy Kinsman, Mairieli Wessel, Marco A. Gerosa, and Christoph Treude. How do software developers use GitHub actions to automate their workflows? In *International Conference on Mining Software Repositories (MSR)*, 2021.
13. Lianping Chen. Continuous delivery: Huge benefits, but challengs too. *IEEE Software - special issue on Release Engineering*, 32(2):50–54, 2015.
14. Daniel Ståhl and Jan Bosch. Experienced benefits of continuous integration in industry software product development: A case study. In *IASTED International Conference on Software Engineering*, pages 736–743, 2013.
15. João Helis Bernardo, Daniel Alencar da Costa, and Uirá Kulesza. Studying the impact of adopting continuous integration on the delivery time of pull requests. In *International Conference on Mining Software Repositories (MSR)*, pages 131–141. IEEE, 2018.

16. Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A. Ernst, and Margaret-Anne Storey. Uncovering the benefits and challenges of continuous integration practices. *IEEE Trans Softw Eng*, 48(7):2570 – 2583, 2022.
17. Lianping Chen. Continuous delivery: Overcoming adoption challenges. *Journal of Systems and Software*, 128:72–86, 2017.
18. Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. An empirical analysis of build failures in the continuous integration workflows of Java-based open-source software. In *International Conference on Mining Software Repositories (MSR)*, pages 345–355. IEEE, 2017.
19. Robin M Betz and Ross C Walker. Implementing continuous integration software in an established computational chemistry software package. In *International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)*, pages 68–74. IEEE, 2013.
20. Jixiang Lu, Zhihong Yang, and Junxia Qian. Implementation of continuous integration and automated testing in software development of smart grid scheduling support system. In *International Conference on Power System Technology*, pages 2441–2446. IEEE, 2014.
21. M Kulas, Jose Luis Borelli, Wolfgang Gässler, Diethard Peter, Sebastian Rabien, Gilles Orban de Xivry, Lorenzo Busoni, Marco Bonaglia, Tommaso Mazzoni, and Gustavo Rahmer. Practical experience with test-driven development during commissioning of the multi-star AO system ARGOS. In *Software and Cyberinfrastructure for Astronomy III*, volume 9152, pages 110–119. SPIE, 2014.
22. Johannes Gmeiner, Rudolf Ramler, and Julian Haslinger. Automated testing in the continuous delivery pipeline: A case study of an online company. In *International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–6. IEEE, 2015.
23. Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. How open source projects use static code analysis tools in continuous integration pipelines. In *International Conference on Mining Software Repositories (MSR)*, pages 334–344. IEEE, 2017.
24. Taher Ahmed Ghaleb, Daniel Alencar Da Costa, and Ying Zou. An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, 24(4):2102–2139, 2019.
25. Carmine Vassallo, Sebastian Proksch, Harald C Gall, and Massimiliano Di Penta. Automated reporting of anti-patterns and decay in continuous integration. In *International Conference on Software Engineering (ICSE)*, pages 105–115. IEEE, 2019.
26. Yash Gupta, Yusaira Khan, Keheliya Gallaba, and Shane McIntosh. The impact of the adoption of continuous integration on developer attraction and retention. In *International Conference on Mining Software Repositories (MSR)*, pages 491–494. IEEE, 2017.
27. David Gray Widder, Michael Hilton, Christian Kästner, and Bogdan 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)*, pages 647–658, 2019.
28. Tingting Chen, Yang Zhang, Shu Chen, Tao Wang, and Yiwen Wu. Let's supercharge the workflows: An empirical study of GitHub Actions. In *International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 01–10. IEEE, 2021.
29. Pablo Valenzuela-Toledo and Alexandre Bergel. Evolution of GitHub Action workflows. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022.
30. Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. On the use of GitHub Actions in software development repositories. In *International Conference on Software Maintenance and Evolution (ICSME)*, 2022.
31. G. Guest, A. Bunce, and L. Johnson. How many interviews are enough? An experiment with data saturation and variability. *Field Methods*, 18(1):59–82, 2006.
32. P. I. Fusch and L. R. Ness. Are we there yet? Data saturation in qualitative research. *The qualitative report*, 20(9), 2015.

33. Armstrong Foundjem, Eleni Constantinou, Tom Mens, and Bram Adams. A mixed-methods analysis of micro-collaborative coding practices in OpenStack. *Empirical Software Engineering*, 27(5):120, 2022.
34. M. Kim, T. Zimmermann, R DeLine, and Begel A. The emerging role of data scientists on software development teams. In *International conference on software engineering (ICSE)*, pages 96–107. IEEE, 2016.
35. A.N. Meyer, E.T. Barr, C Bird, and T . Zimmermann. Today was a good day: The daily life of software developers. *IEEE Trans Softw Eng*, 47(5):863–880, 2019.
36. H. Russel Bernard, Amber Wutich, and Gery W. Ryan. *Analyzing qualitative data: Systematic approaches.* SAGE publications, 2nd edition, 2016.
37. Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. Trade-offs in continuous integration: assurance, security, and flexibility. In *Joint Meeting on Foundations of Software Engineering(FSE)*, pages 197–207, 2017.
38. Martin Fowler and Matthew Foemmel. Continuous integration, 2006.
39. Marko Leppänen, Simo Mäkinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V Mäntylä, and Tomi Männistö. The highways and country roads to continuous deployment. *IEEE Software*, 32(2):64–72, 2015.
40. Akond Rahman, Amritanshu Agrawal, Rahul Krishna, and Alexander Sobran. Characterizing the influence of continuous integration: Empirical results from 250+ open source and proprietary projects. In *ACM SIGSOFT International Workshop on Software Analytics*, pages 8–14, 2018.
41. Carmine Vassallo, Fabio Palomba, and Harald C Gall. Continuous refactoring in CI: A preliminary study on the perceived advantages and barriers. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 564–568. IEEE, 2018.
42. Zhi Li, Weijie Liu, Hongbo Chen, X Wang, Xiaojing Liao, Luyi Xing, Mingming Zha, Hai Jin, and Deqing Zou. Robbery on DevOps: Understanding and mitigating illicit cryptomining on continuous integration service platforms. In *IEEE Symposium on Security and Privacy (SP)*, pages 2397–2412. IEEE, 2022.
43. Mendy, Montana and Rios, Nicolas and Rybinski, Michal. The new pricing model for travis-ci.com. `https://blog.travis-ci.com/2020-11-02-travis-ci-new-billing`, 2020. Accessed: 14.10.2022.
44. Machiraju, Vijay. Change in Azure pipelines grant for public projects. `https://devblogs.microsoft.com/devops/change-in-azure-pipelines-grant-for-public-projects/`, 2021. Accessed: 14.10.2022.
45. Hamed Esfahani, Jonas Fietz, Qi Ke, Alexei Kolomiets, Erica Lan, Erik Mavrinac, Wolfram Schulte, Newton Sanches, and Srikanth Kandula. CloudBuild: Microsoft's distributed and caching build service. In *International Conference on Software Engineering (ICSE)*, pages 11–20, 2016.
46. Keheliya Gallaba, Yves Junqueira, John Ewart, and Shane Mcintosh. Accelerating continuous integration by caching environments and inferring dependencies. *IEEE Trans Softw Eng*, 48(6):2040–2052, 2022.
47. Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. Which commits can be CI skipped? *IEEE Trans Softw Eng*, 47(3):448–463, 2019.
48. Rabe Abdalkareem, Suhaib Mujahid, and Emad Shihab. A machine learning approach to improve the detection of CI skip commits. *IEEE Trans Softw Eng*, 2020.
49. Xianhao Jin and Francisco Servant. A cost-efficient approach to building in continuous integration. In *International Conference on Software Engineering (ICSE)*, pages 13–25. IEEE, 2020.
50. Xianhao Jin and Francisco Servant. What helped, and what did not? An evaluation of the strategies to improve continuous integration. In *International Conference on Software Engineering (ICSE)*, pages 213–225. IEEE, 2021.
51. Xianhao Jin and Francisco Servant. Which builds are really safe to skip? Maximizing failure observation for build selection in continuous integration. *Journal of Systems and Software*, 188, 2022.
52. Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology*, 106:201–230, 2019.

53. Paul Ralph and Ewan Tempero. Construct validity in software engineering research and software metrics. In *International Conference on Evaluation and Assessment in Software Engineering*, pages 13–23, 2018.
54. Joseph Maxwell. Understanding and validity in qualitative research. *Harvard educational review*, 62(3):279–301, 1992.

A Interview Questionnaire

The interview questions were structured in 6 categories. Some questions were conditional to the responses received on previous questions:

1. General questions about the respondent:
   a) Please briefly introduce yourself.
   b) Report on your past and current experience in collaborative software development, and on the kinds of projects you are or have been actively involved in, for which CI/CD tools have been used.
   c) What is or was your involvement in those projects?
   d) How many years of experience do you have with CI/CD?
2. General questions about CI/CD usage:
   a) When did you *first start to use* a CI/CD tool in those projects and what was the reason at that time?
   b) What are *currently* the main reasons for using CI/CD in those projects?
3. Questions about specific CI/CD tool usage:
   a) Which different CI/CD tools have you used in the past, or are you currently using?
   b) Why did you or the project maintainers decide to use that particular CI/CD tool?
   c) What are the resources (in terms of budget, hardware, personnel, etc.) and effort that are or were available and required for creating, hosting and maintaining the CI/CD infrastructure for your projects?
   d) *[If one of the reported CI/CD tools was Travis:]*
      – Was Travis a kind of default choice, or was it a deliberate choice?
      – Are you aware of Travis ' changes in its free plan for public repositories? Has your project been affected by these changes?
   e) *[If none of the reported CI/CD tools was Travis or GitHub Actions:]*
      Why haven't you ever used Travis or GitHub Actions?
   f) What were the main reasons for using these CI/CD tools, and what were/are the advantages and shortcomings of each of them according to your experience?
4. Questions about CI/CD migration: [These questions should be answered for every project that was reported by the respondent.]
   a) Did the project migrate from some CI/CD tool to another one during its lifetime?
   b) *[In case of negative answer to 4.a:]*
      Even if the project did not migrate its CI/CD tool, did you ever consider migrating to another CI/CD tool? If yes, why didn't you carry out the migration?
   c) *[In case of positive answer to 4.a:]*
      – When did the project perform the migration?
      – From which CI/CD tool to which other CI/CD tool?
      – What drove the decision to migrate, and on which replacement CI/CD tool to adopt? (Was the migration because you disliked something in the existing CI/CD tool? Or because you liked something better in the replacement CI/CD tool?)
      – How much effort and time did it take to do the CI/CD migration and why?
      – What were the main difficulties (if any) in doing the migration?
      – How satisfied were you with the replacement CI/CD tool?
   d) *[In case the respondent did not mention GitHub Actions as a CI/CD migration target:]*
      – Are you aware of GitHub Actions and its increasing popularity? Why do you think this is the case?
      – Did you ever consider using GitHub Actions for doing CI/CD?
      – If not, why not? What is missing in GitHub Actions in order for the project to migrate to it?
   e) To what extent has the acquisition of GitHub by Microsoft in June 2018 affected you? Did it trigger you to migrate from one platform to another, for example from Github to GitLab or vice versa?
5. Questions about CI/CD tool co-usage:
   a) Did or does the same project use multiple different CI/CD tools simultaneously? Which ones?

   b) When and for how long have they been used together?
   c) What is or was the reason for using multiple CI/CD tools within the same project? What is or was the purpose of each CI/CD tool?
6. Closing open-ended question:
   a) Do you have any other important remarks related to CI/CD tool usage that you would like to share with us?

B Mapping between respondents and CI/CD tools

Throughout the article we have used respondent IDs whenever we cited relevant quotes from the interviews conducted with them. In order to put these quotes in the right perspective, the table below provides a mapping between the respondent IDs and the CI/CD tools that these respondents mentioned to have used somewhere during their career.

Table 8 Mapping between CI/CD tools and respondents having reported to use them.

CI tool	**respondent IDs**
GHA	1 2 4 5 6 8 9 10 12 13 14 15 17 18 19 20 21 22
Jenkins	1 2 4 5 7 8 11 12 13 14 15 16 17 18 21 22
Travis	2 4 5 8 9 10 11 12 13 16 18 19 20 21 22
GitLab CI/CD	1 2 3 4 5 7 8 11 12 13 14 15 16 22
CircleCI	2 4 6 8 9 10 12 13 18 19 20 22
Azure DevOps	2 4 5 9 12 14 15 17 18 19 20
AppVeyor	9 13 18 19 22
Hudson	4 5 6 15 22
TeamCity	1 7 13
Bamboo	2 4
Bitbucket Pipelines	7 20
Cruise Control	4 6
Drone	9 22
Netlify	10 14
AWS CI/CD	5
Buildbot	18
BuildKite	13
Cirrus CI	19
Codefresh	5
Concourse	5
Heroku	8
Jacamar CI	4
Percy	8
Pulumi	5
Sauce Labs	21
Tekton	4
Vercel	10
Zuul	19
custom-built in-house solution	6, 12, 18