Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects

Michael Hilton
Oregon State University, USA
hiltonm@eecs.oregonstate.edu

Timothy Tunnell
University of Illinois, USA
tunnell2@illinois.edu

Kai Huang
University of Illinois, USA
khuang29@illinois.edu

Darko Marinov
University of Illinois, USA
marinov@illinois.edu

Danny Dig
Oregon State University, USA
digd@eecs.oregonstate.edu

ABSTRACT

Continuous integration (CI) systems automate the compilation, building, and testing of software. Despite CI rising as a big success story in automated software engineering, it has received almost no attention from the research community. For example, how widely is CI used in practice, and what are some costs and benefits associated with CI? Without answering such questions, developers, tool builders, and researchers make decisions based on folklore instead of data.

In this paper, we use three complementary methods to study the usage of CI in open-source projects. To understand *which* CI systems developers use, we analyzed 34,544 open-source projects from GitHub. To understand *how* developers use CI, we analyzed 1,529,291 builds from the most commonly used CI system. To understand *why* projects use or do not use CI, we surveyed 442 developers. With this data, we answered several key questions related to the usage, costs, and benefits of CI. Among our results, we show evidence that supports the claim that CI helps projects release more often, that CI is widely adopted by the most popular projects, as well as finding that the overall percentage of projects using CI continues to grow, making it important and timely to focus more research on CI.

CCS Concepts

•**Software and its engineering** → *Agile software development;* **Software testing and debugging;**

Keywords

continuous integration; mining software repositories

1. INTRODUCTION

Continuous Integration (CI) is emerging as one of the biggest success stories in automated software engineering. CI systems automate the compilation, building, testing and deployment of software. For example, such automation has been reported [22] to help Flickr deploy to production more than 10 times per day. Others [40] claim that by adopting CI and a more agile planning process, a product group at HP reduced development costs by 78%.

These success stories have led to CI growing in interest and popularity. Travis CI [17], a popular CI service, reports that over 300,000 projects are using Travis. The State of Agile industry survey [48], with 3,880 participants, found 50% of respondents use CI. The State of DevOps report [49] finds CI to be one of the indicators of "high performing IT organizations". Google Trends [11] shows a steady increase of interest in CI: searches for "Continuous Integration" increased 350% in the last decade.

Despite the growth of CI, the only published research paper related to CI usage [53] is a preliminary study, conducted on 246 projects, which compares several quality metrics of projects that use or do not use CI. However, the study does not present any detailed information on *how* projects use CI. In fact, despite some folkloric evidence about the use of CI, there is no systematic study about CI systems.

Not only do we lack basic knowledge about the extent to which open-source projects are adopting CI, but also we have no answers to many important questions related to CI. What are the costs of CI? Does CI deliver on the promised benefits, such as releasing more often, or helping make changes (e.g., to merge pull requests) faster? Do developers maximize the usage of CI? Despite the widespread popularity of CI, we have very little quantitative evidence on its benefits. This lack of knowledge can lead to poor decision making and missed opportunities. Developers who choose not to use CI can be missing out on the benefits of CI. Developers who do choose to use CI might not be using it to its fullest potential. Without knowledge of how CI is being used, tool builders can be misallocating resources instead of having data about where automation and improvements are most needed by their users. By not studying CI, researchers have a blind spot which prevents them from providing solutions to the hard problems that practitioners face.

In this paper we use three complementary methods to study the usage of CI in open-source projects. To understand the extent to which CI has been adopted by developers, and *which* CI systems developers use, we analyzed 34,544 open-source projects from GitHub. To understand *how* developers use CI, we analyzed 1,529,291 builds from Travis CI, the most commonly used CI service for GitHub projects (Section 4.1).

ASE'16, September 3–7, 2016, Singapore, Singapore
© 2016 ACM. 978-1-4503-3845-5/16/09...
http://dx.doi.org/10.1145/2970276.2970358

To understand *why* projects use or do not use CI, we surveyed 442 developers.

With this data, we answer several research questions that we grouped into three themes:

Theme 1: *Usage of CI*
RQ1: *What percentage of open-source projects use CI?*
RQ2: *What is the breakdown of usage of different CI services?*
RQ3: *Do certain types of projects use CI more than others?*
RQ4: *When did open-source projects adopt CI?*
RQ5: *Do developers plan on continuing to use CI?*
We found that CI is widely used, and the number of projects which are adopting CI is growing. We also found that the most popular projects are most likely to use CI.

Theme 2: *Costs of CI*
RQ6: *Why do open-source projects choose* not *to use CI?*
RQ7: *How often do projects evolve their CI configuration?*
RQ8: *What are some common reasons projects evolve their CI configuration?*
RQ9: *How long do CI builds take on average?*
We found that the most common reason why developers are not using CI is lack of familiarity with CI. We also found that the average project makes only 12 changes to their CI configuration file and that many such changes can be automated.

Theme 3: *Benefits of CI*
RQ10: *Why do open-source projects choose to use CI?*
RQ11: *Do projects with CI release more often?*
RQ12: *Do projects which use CI accept more pull requests?*
RQ13: *Do pull requests with CI builds get accepted faster (in terms of calendar time)?*
RQ14: *Do CI builds fail less on master than on other non-master branches?*
We first surveyed developers about the perceived benefits of CI, then we empirically evaluated these claims. We found that projects that use CI release twice as often as those that do not use CI. We also found that projects with CI accept pull requests faster than projects without CI.

This paper makes the following contributions:

1. **Research Questions:** We designed 14 novel research questions. We are the first to provide in-depth answers to questions about the usage, costs, and benefits of CI.

2. **Data Analysis:** We collected and analyzed CI usage data from 34,544 open-source projects. Then we analyzed in-depth all CI data from a subset of 620 projects and their 1,529,291 builds, 1,503,092 commits, and 653,404 pull requests. Moreover, we surveyed 442 open-source developers about why they chose to use or not use CI.

3. **Implications:** We provide practical implications of our findings from the perspective of three audiences: researchers, developers, and tool builders. Researchers should pay attention to CI because it is not a passing fad. For developers we list several situations where CI provides the most value. Moreover, we discovered several opportunities where automation can be helpful for tool builders.

More details about our data sets and results are available at http://cope.eecs.oregonstate.edu/CISurvey

2. OVERVIEW OF CI

2.1 History and Definition of CI

The idea of Continuous Integration (CI) was first introduced in 1991 by Grady Booch [26], in the context of object-oriented design: "At regular intervals, the process of *continuous integration* yields executable releases that grow in functionality at every release..." This idea was then adopted as one of the core practices of Extreme Programming (XP) [23].

However, the idea began to gain acceptance after a blog post by Martin Fowler [37] in 2000. The motivating idea of CI is that the more often a project can integrate, the better off it is. The key to making this possible, according to Fowler, is automation. Automating the build process should include retrieving the sources, compiling, linking, and running automated tests. The system should then give a "yes" or "no" indicator of whether the build was successful. This automated build process can be triggered either manually or automatically by other actions from the developers, such as checking in new code into version control.

These ideas were implemented by Fowler in CruiseControl [9], the first CI system, which was released in 2001. Today there are over 40 different CI systems, and some of the most well-known ones include Jenkins [12] (previously called Hudson), Travis CI [17], and Microsoft Team Foundation Server (TFS) [15]. Early CI systems usually ran locally, and this is still widely done for Jenkins and TFS. However, CI as a service has become more and more popular, e.g., Travis CI is only available as a service, and even Jenkins is offered as a service via the CloudBees platform [6].

2.2 Example Usage of CI

We now present an example of CI that comes from our data. The pull request we are using can be found here: https://github.com/RestKit/RestKit/pull/2370. A developer named "Adlai-Holler" created pull request #2370 named "Avoid Flushing In-Memory Managed Object Cache while Accessing" to work around an issue titled "Duplicate objects created if inserting relationship mapping using RKInMemoryManagedObjectCache" for the project *RestKit* [13]. The developer made two commits and then created a pull request, which triggered a Travis CI build. The build failed, because of failing unit tests. A RestKit project member,"segiddins", then commented on the pull request, and asked Adlai-Holler to look into the test failures. Adlai-Holler then committed two new changes to the same pull request. Each of these commits triggered a new CI build. The first build failed, but the second was successful. Once the CI build passed, the RestKit team member commented "seems fine" and merged the pull request.

3. METHODOLOGY

To understand the extent to which CI is used and *which* CI systems developers use, we analyzed 34,544 open-source projects from GitHub with our *breadth corpus*. To understand *how* developers use CI, we analyzed 1,529,291 builds on the most popular CI system in our *depth corpus*. To understand *why* projects use or do not use CI, we surveyed 442 developers.

3.1 Breadth Corpus

The breadth corpus has a large number of projects, and information about what CI services each project uses. We use the breadth corpus to answer broad questions about the usage of CI in open-source projects. We collected the data for this corpus primarily via the GitHub API. We first sorted GitHub projects by their popularity, using the star rating (whereby users can mark, or "star", some projects that they like, and hence each project can accumulate stars). We started our inspection from the top of the list, first by manually looking at the top 50 projects. We collected all publicly available information about how these projects use CI. We then used what we learned from this manual inspection to write a script to programmatically classify which CI service (if any) a project uses. The four CI services that we were able to readily identify manually and later by our script are (sorted in the order of their usage): Travis CI [17], CircleCI [5], AppVeyor [2], and Werker [18]. All of these services provide public API's which we queried to determine if a project is using that service.

Moreover, we wanted to ensure that we had collected as complete data as possible. When we examined the data by hand, we found that several projects were using CloudBees [6], a CI service powered by the Jenkins CI. However, given a list of GitHub projects, there is no reliable way to programmatically identify from the GitHub API which projects use CloudBees. (In contrast, Travis CI uses the same organization and project names as GitHub, making it easy to check correspondence between Travis CI and GitHub projects.) We contacted CloudBees, and they sent us a list of open-source projects that have CloudBees build set up. We then wrote a script to parse that list, inspect the build information, and search for the corresponding GitHub repository (or repositories) for each build on CloudBees. We then used this data to identify the projects from our breadth corpus that use CloudBees. This yielded 1,018 unique GitHub repositories/projects. To check whether these projects refer to CloudBees, we searched for (case insensitive) "CloudBees" in the README files of these projects and found that only 256 of them contain "CloudBees". In other words, had we not contacted CloudBees directly, using only the information available on GitHub, we would have missed a large number of projects that use CloudBees.

Overall, the breadth corpus consists of 34,544 projects. For each project, we collected the following information: project name and owner, the CI system(s) that the project uses (if any), popularity (as measured by the number of stars), and primary programming language (as determined by GitHub).

3.2 Depth Corpus

The depth corpus has fewer projects, but for each project we collect all the information that is publicly available. For this subset of projects, we collected additional data to gain a deeper understanding of the usage, costs, and benefits of CI. Analyzing our breadth corpus, as discussed in Section 4.1, we learned that Travis CI is by far the most commonly used CI service among open-source projects. Therefore, we targeted projects using Travis CI for our depth corpus. First, we collected the top 1,000 projects from GitHub ordered by their popularity. Of those 1,000 projects, we identified 620 projects that use Travis CI, 37 use AppVeyor, 166 use CircleCI, and 3 use Werker. We used the Travis CI API[1] to collect the entire build history for each project in our depth corpus, for a total of 1,529,291 builds. Using GHTorrent [39], we collected the full history of pull requests for each project, for a total of 653,404 pull requests. Additionally, we cloned every project in our corpus, to access the entire commit history and source code.

3.3 Survey

Even after collecting our diverse breadth and depth corpora, we were still left with questions that we could not answer from the online data alone. These questions were about *why* developers chose to use or not use CI. We designed a survey to help us answer a number of such "why" questions, as well as to provide us another data source to better understand CI usage. We deployed our survey by sending it to all the email addresses publicly listed as belonging to the organizations of all the top 1,000 GitHub projects (again rated by the popularity). In total, we sent 4,508 emails.

Our survey consisted of two flows, each with three questions. The first question in both flows asked if the participant used CI or not. Depending on the answer they gave to this question, the second question asked the reasons why they use or do not use CI. These questions were multiple-choice, multiple-selection questions where the users were asked to select all the reasons that they agreed with. To populate the choices, we collected some common reasons for using or not using CI, as mentioned in websites [1, 7], blogs [3, 8, 19], and Stack Overflow [14]. Optionally, the survey participants could also write their own reason(s) that we did not already list. The third question asked if the participant plans on using CI for future projects.

To incentivize participation, we raffled off a 50 USD gift card among the survey respondents. 442 (9.8% response rate) participants responded to our survey. Of those responses, 407 (92.1%) indicated that they do use CI, and 35 (7.9%) indicated that they do not use CI.

4. RESULTS

In this section, we present the results to our research questions. Section 4.1 presents the results about the usage of CI. Section 4.2 discusses the costs of CI. Finally Section 4.3 presents the benefits of CI. Rather than presenting implications after each research question, we draw from several research questions to triangulate implications that we present in Section 5.

4.1 Usage of CI

To determine the extent to which CI is used, we study what percentage of projects actively use CI, and we also ask developers if they plan to use CI in the future. Furthermore, we study whether the project popularity and programming language correlate with the usage of CI.

RQ1: *What percentage of open-source projects use CI?*

We found that 40% of all the projects in our breadth corpus use CI. Table 1 shows the breakdown of the usage. Thus, CI is indeed used widely and warrants further investigation.

[1]We are grateful to the Travis CI developers for promptly resolving a bug report that we submitted; prior to them resolving this bug report, one could not query the full build history of all projects.

Table 1: Breadth corpus projects CI usage

Project Uses CI?	Percentage	Number of Projects
Yes	40.27%	13,910
No	59.73%	20,634

Additionally, we know that our scripts do not find all CI usage (e.g., projects that run privately hosted CI systems, as discussed further in Section 6.2). We can reliably detect the use of (public) CI services only if their API makes it possible to query the CI service based on knowing the GitHub organization and project name. Therefore, the results we present are a lower bound on the total number of projects that use CI.

Table 2: CI usage by Service. The top row shows percent of all CI projects using that service, the second row shows the total number of projects for each service. Percents add up to more than 100 due to some projects using multiple CI services.

Usage by CI Service				
Travis	CircleCI	AppVeyor	CloudBees	Werker
90.1%	19.1%	3.5%	1.6%	0.4%
12528	2657	484	223	59

RQ2: *What is the breakdown of usage of different CI services?*

Next we investigate which CI services are the most widely used in our breadth corpus. Table 2 shows that Travis CI is by far the most widely used CI service. Because of this result, we feel confident that our further analysis can focus on the projects that use Travis CI as a CI service, and that analyzing such projects gives representative results for usage of CI services in open-source projects.

We also found that some projects use more than one CI service. In our breadth corpus, of all the projects that use CI, 14% use more than one CI. We think this is an interesting result which deserves future attention.

RQ3: *Do certain types of projects use CI more than others?*

To better understand which projects use CI, we look for characteristics of projects that are more likely to use CI.

CI usage by project popularity: We want to determine whether more popular projects are more likely to use CI. Our intuition is that if CI leads to better outcomes, then we would expect to see higher usage of CI among the most popular projects (or, alternatively, that projects using CI get better and thus are more popular). Figure 1 shows that the most popular projects (as measured by the number of stars) are also the most likely to use CI (Kendall's τ, $p < 0.00001$). We group the projects from our breadth corpus into 64 even groups, ordered by number of stars. We then calculate the percent of projects in each group that are using CI. Each group has around 540 projects. In the most popular (starred) group, 70% of projects use CI. As the projects become less popular, the percentage of projects using CI declines to 23%.

Observation

Popular projects are more likely to use CI.

CI usage by language: We now examine CI usage by programming language. Are there certain languages for which the projects written primarily in such languages use CI more than others? Table 3 shows projects sorted by the percentage of projects that use CI for each language, from our breadth corpus. The data shows that in fact there are certain languages that use CI more than others. Notice that the usage of CI does not perfectly correlate with the number of projects using that language (as measured by the number of projects using a language, with its rank by percentage, Kendall's τ, $p > 0.68$). In other words, some of the languages that use CI the most are both popular languages like Ruby and emerging languages like Scala. Similarly, among projects that use CI less, we notice both popular languages such as Objective-C and Java, as well as less popular languages such as VimL.

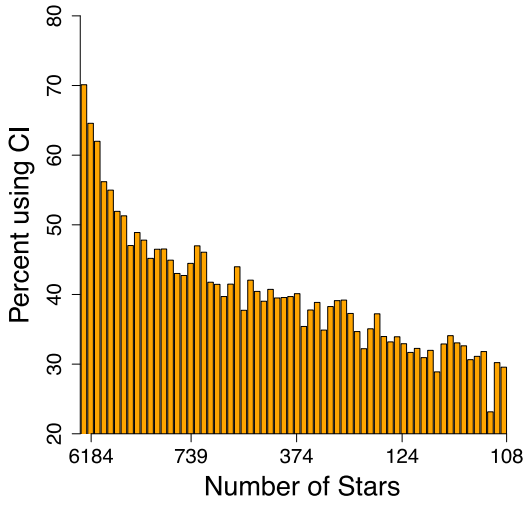

Figure 1: CI usage of projects in GitHub. Projects are sorted by popularity (number of stars).

However, we did observe that many of the languages that have the highest CI usage are also dynamically-typed languages (e.g., Ruby, PHP, CoffeeScript, Clojure, Python, and JavaScript). One possible explanation may be that in the absence of a static type system which can catch errors early on, these languages use CI to provide extra safety.

Observation

We observe a wide range of projects that use CI. The popularity of the language does not correlate with the probability that a project uses CI.

RQ4: *When did open-source projects adopt CI?*

We next study when projects began to adopt CI. Figure 2 shows the number of projects using CI over time. We answer this question with our depth corpus, because the breadth corpus does not have the date of the first build, which we use to determine when CI was introduced to the project. Notice that we are collecting data from Travis CI, which was founded in 2011 [10]. Figure 2 shows that CI has experienced a steady growth over the last 5 years.

We also analyze the age of each project when developers first introduced CI, and we found that the median time was around 1 year. Based on this data, we conjecture that while many developers introduce CI early in a project's

Table 3: CI usage by programing language. For each language, the columns tabulate: the number of projects from our corpus that predominantly use that language, how many of these projects use CI, the percentage of projects that use CI.

Language	Total Projects	# Using CI	Percent CI
Scala	329	221	67.17
Ruby	2721	1758	64.61
Go	1159	702	60.57
PHP	1806	982	54.37
CoffeeScript	343	176	51.31
Clojure	323	152	47.06
Python	3113	1438	46.19
Emacs Lisp	150	67	44.67
JavaScript	8495	3692	43.46
Other	1710	714	41.75
C++	1233	483	39.17
Swift	723	273	37.76
Java	3371	1188	35.24
C	1321	440	33.31
C#	652	188	28.83
Perl	140	38	27.14
Shell	709	185	26.09
HTML	948	241	25.42
CSS	937	194	20.70
Objective-C	2745	561	20.44
VimL	314	59	18.79

development lifetime, it is not always seen as something that provides a large amount of value during the very initial development of a project.

Observation

The median time for CI adoption is one year.

RQ5: *Do developers plan on continuing to use CI?* Is CI a passing "fad" in which developers will lose interest, or will it be a lasting practice? While only time will tell what the true answer is, to get some sense of what the future could hold, we asked developers in our survey if they plan to use CI for their next project. We asked them how likely they were to use CI on their next project, using a 5-point Likert scale ranging from definitely will use to definitely will not use. Figure 3 shows that developers feel very strongly that they will be using CI for their next project. The top two options, 'Definitely' and 'Most Likely', account for 94% of all our survey respondents, and the average of all the answers was 4.54. While this seems like a pretty resounding endorsement for the continued use of CI, we decided to dig a little deeper. Even among respondents who are not currently using CI, 53% said that they would 'Definitely' or 'Most Likely' use CI for their next project.

Observation

While CI is widely used in practice nowadays, we predict that in the future, CI adoption rates will increase even further.

4.2 Costs of CI

To better understand the costs of CI, we analyze both the survey (where we asked developers why they believe CI is too costly to be worth using) and the data from our depth

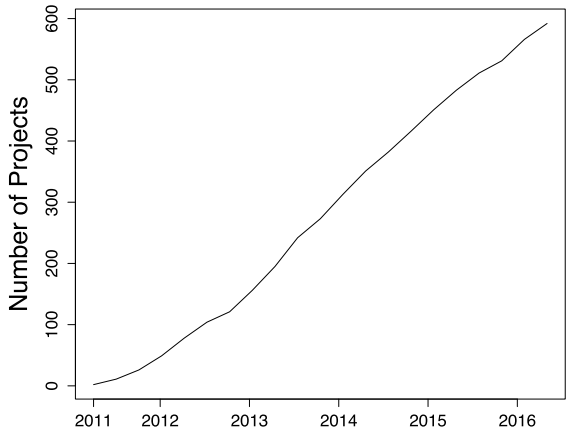

Figure 2: Number of projects using CI over time. Data is tabulated by quarter (3 months) per year.

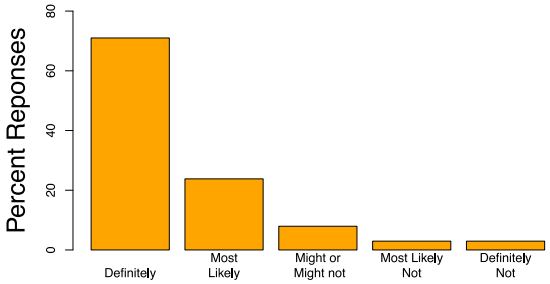

Figure 3: Answers to "Will you use CI for your next project?"

corpus. We estimate the cost to developers for writing and maintaining the configuration for their CI service. Specifically, we measure how often the developers make changes to their configuration files and study why they make those changes to the configuration files. We also analyze the cost in terms of the time to run CI builds. Note that the time that the builds take to return a result could be unproductive time if the developers do not know how to proceed without knowing that result.

RQ6: *Why do open-source projects choose* not *to use CI?*

One way to evaluate the costs of CI is to ask developers why they do *not* use CI. In our survey, we asked respondents whether they chose to use or not use CI, and if they indicated that they did not, then we asked them to tell us *why* they do not use CI.

Table 4 shows the percentage of the respondents who selected particular reasons for not using CI. As mentioned before, we built the list of possible reasons by collecting information from various popular internet sources. Interestingly, the primary cost that respondents identified was not a technical cost; instead, the reason for not using CI was that *"The developers on my project are not familiar enough with CI."* We do not know if the developers are not familiar enough with the CI tools themselves (e.g., Travis CI), or if they are unfamiliar with all the work it will take to add CI to their project, including perhaps fully automating the build. To completely answer this question, more research is needed.

The second most selected reason was that the project does not have automated tests. This speaks to a real cost for CI, in

Table 4: Reasons developers gave for not using CI

Reason	Percent
The developers on my project are not familiar enough with CI	47.00
Our project doesn't have automated tests	44.12
Our project doesn't commit often enough for CI to be worth it	35.29
Our project doesn't currently use CI, but we would like to in the future	26.47
CI systems have too high maintenance costs (e.g., time, effort, etc.)	20.59
CI takes too long to set up	17.65
CI doesn't bring value because our project already does enough testing	5.88

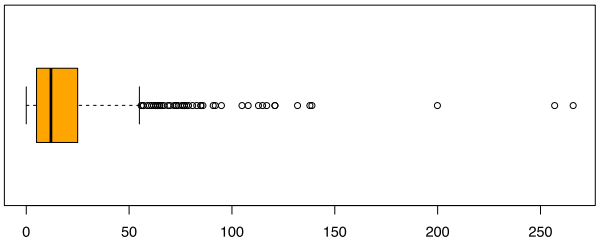

Figure 4: Number of changes to CI configs, median number of changes is 12

that much of its value comes from automated tests, and some projects find that developing good automated test suites is a substantial cost. Even in the cases where developers had automated tests, some questioned the use of CI (in particular and regression testing in general); one respondent (P74) even said *"In 4 years our tests have yet to catch a single bug."*

Observation

The main reason why open-source projects choose to not use CI is that the developers are not familiar enough with CI.

RQ7: *How often do projects evolve their CI configuration?* We ask this question to identify how often developers evolve their CI configurations. Is it a "write-once-and-forget-it" situation, or is it something that evolves constantly? The Travis CI service is configured via a YAML [20] file, named `.travis.yml`, in the project's root directory. YAML is a human-friendly data serialization standard. To determine how often a project has changed its configuration, we analyzed the history of every `.travis.yml` file and counted how many times it has changed. We calculate the number of changes from the commits in our depth corpus. Figure 4 shows the number of changes/commits to the `.travis.yml` file over the life of the project. We observe that the median of number of changes to a project's CI configuration is 12 times, but one of the projects changed the CI configuration 266 times. This leads us to conclude that many projects setup CI once and then have minimal involvement (25% of projects have 5 or less changes to their CI configuration), but some projects do find themselves changing their CI setup quite often.

Observation

Some projects change their configurations relatively often, so it is worthwhile to study what these changes are.

Table 5: Reasons for CI config changes

Config Area	Total Edits	Percentage
Build Matrix	9718	14.70
Before Install	8549	12.93
Build Script	8328	12.59
Build Language Config	7222	10.92
Build Env	6900	10.43
Before Build Script	6387	9.66
Install	4357	6.59
Whitespace	3226	4.88
Build platform Config	3058	4.62
Notifications	2069	3.13
Comments	2004	3.03
Git Configuration	1275	1.93
Deploy Targets	1079	1.63
After Build Success	1025	1.55
After Build Script	602	0.91
Before Deploy	133	0.20
After Deploy	79	0.12
Custom Scripting	40	0.06
After Build Failure	39	0.06
After Install	14	0.02
Before Install	10	0.02
Mysql	5	0.01
After Build Success	3	0.00
Allow Failures	2	0.00

RQ8: *What are some common reasons projects evolve their CI configuration?*

To better understand the changes to the CI configuration files, we analyzed all the changes that were made to the `.travis.yml` files in our depth corpus. Because YAML is a structured language, we can parse the file and determine which part of the configuration was changed. Table 5 shows the distribution of all the changes. The most common changes were to the *build matrix*, which in Travis specifies a combination of *runtime*, *environment*, and *exclusions/inclusions*. For example, a build matrix for a project in Ruby could specify the runtimes *rvm 2.2*, *rvm 1.9*, and *jruby*, the build environment *rails2* and *rails3*, and the exclusions/inclusions, e.g., *exclude: jruby with rails2*. All combinations will be built except those excluded, so in this example there would be 5 different builds. Other common changes included the dependent libraries to install before building the project (what `.travis.yml` calls *before install*) and changes to the build script themselves. Also, many other changes were due to the version changes of dependencies.

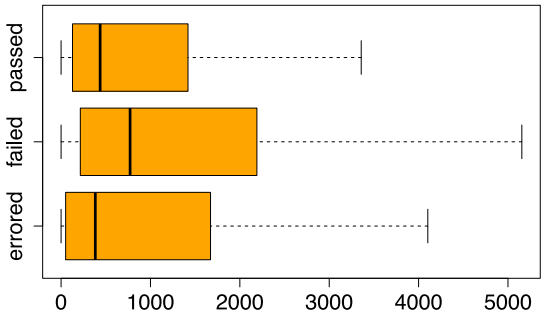

Figure 5: Build time distribution by result, in seconds

Observation

Many CI configuration changes are driven by dependency changes and could be potentially automated.

RQ9: *How long do CI builds take on average?*

Another cost of using CI is the time to build the application and run all the tests. This cost represents both a cost of energy[2] for the computing power to run these builds, but also developers may have to wait to see if their build passes before they merge in the changes, so having longer build times means more wasted developer time.

The average build time is just under 500 seconds. To compute the average build times, we first remove all the canceled (incomplete, manually stopped) build results, and only consider the time for errored, failed, and passed (completed builds). Errored builds are those that occur before the build begins (e.g., when a dependency cannot be downloaded), and failed builds are those that the build is not completed succesfully (e.g., a unit test fail). To further understand the data, we look at each outcome independently. Interestingly, we find that passing builds run faster than either errored or failed builds. The difference between errored and failed is significant (Wilcoxon, $p < 0.0001$), as is the difference between passed and errored (Wilcoxon, $p < 0.0001$) and the difference between passed and failed (Wilcoxon, $p < 0.0001$).

We find this result surprising as our intuition is that passing builds should take longer, because if an error state is encountered early on, the process can abort and return earlier. Perhaps it is the case that many of the faster running pass builds are not generating a meaningful result, and should not have been run. However, more investigation is needed to determine what the exact reasons for this are.

4.3 Benefits of CI

We first summarize the most commonly touted benefits of CI, as reported by the survey participants. We then analyze empirically whether these benefits are quantifiable in our depth corpus. Thus, we confirm or refute previously held beliefs about the benefits of CI.

[2]This cost should not be underestimated; our personal correspondence with a Google manager in charge of their CI system TAP reveals that TAP costs millions of dollars just for the computation (not counting the cost of developers who maintain or use TAP).

RQ10: *Why do open-source projects choose to use CI?*

Having found that CI is widely used in open-source projects (RQ1), and that CI is most widely used among the most popular projects on GitHub (RQ3), we want to understand why developers choose to use CI. However, *why* a project uses CI cannot be determined from a code repository. Thus, we answer this question using our survey data.

Table 6 shows the percentage of the respondents who selected particular reasons for using CI. As mentioned before, we build this list of reasons by collecting information from various popular internet sources. The two most popular reasons were "CI makes us less worried about breaking our builds" and "CI helps us catch bugs earlier". One respondent (P371) added: *"Acts like a watchdog. You may not run tests, or be careful with merges, but the CI will. :)"*

Martin Fowler [7] is quoted as saying "Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove." However, in our survey, very few projects felt that CI actually helped them during the debugging process.

Observation

Projects use CI because it helps them catch bugs early and makes them less worried about breaking the build. However, CI is not widely perceived as helpful with debugging.

RQ11: *Do projects with CI release more often?*

One of the more common claims about CI is that it helps projects release more often, e.g., CloudBees motto is "Deliver Software Faster" [6]. Over 50% of the respondents from our survey claimed it was a reason why they use CI. We analyze our data to see if we can indeed find evidence that would support this claim.

We found that projects that use CI do indeed release more often than either (1) the same projects before they used CI or (2) the projects that do not use CI. In order to compare across projects and periods, we calculated the release rate as the number of releases per month. Projects that use CI average .54 releases per month, while projects that do not use CI average .24 releases per month. That is more than double the release rate, and the difference is statistically significant (Wilcoxon, $p < 0.00001$). To identify the effect of CI, we also compared, for projects that use CI, the release rate both before and after the first CI build. We found that projects that eventually added CI used to release at a rate of .34 releases per month, well below the .54 rate at which they release now with CI. This difference is statistically significant (Wilcoxon, $p < 0.00001$).

Observation

Projects that use CI release more than twice as often as those that do not use CI.

RQ12: *Do projects which use CI accept more pull requests?*

For a project that uses a CI service such as Travis CI, when the CI server builds a pull request, it annotates the pull request on GitHub with a visual cue such as a green check mark or a red 'X' that shows whether the pull request was able to build successfully on the CI server. Our intuition is that this extra information can help developers better decide whether or not to merge a pull request into their code. To determine if this extra information indeed makes a difference, we compared the pull request acceptance rates between pull

Table 6: Reasons for using CI, as reported by survey participants

Reason	Percent
CI makes us less worried about breaking our builds	87.71
CI helps us catch bugs earlier	79.61
CI allows running our tests in the cloud, freeing up our personal machines	54.55
CI helps us deploy more often	53.32
CI makes integration easier	53.07
CI runs our tests in a real-world staging environment	46.00
CI lets us spend less time debugging	33.66

Table 7: Release rate of projects

Uses Travis	Versions Released per Month
Yes	.54
No	.24

Table 8: Comparison of pull requests merged for pull requests that had or did not have CI information

CI Usage	% Pull Requests Merged
Using CI	23
Not Using CI	28

requests that have this CI information and pull requests that do not have it, from the depth corpus. Note that projects can exclude some branches from their repository to not run on the CI server, so just because a project uses CI on some branch, there is no guarantee that every pull request contains the CI build status information.

Table 8 shows the results for this question. We found that pull requests without CI information were 5pp more likely to be merged than pull requests with CI information. Our intuition of this result is that those 5pp of pull requests have problems which are identified by the CI. By not merging these pull requests, developers can avoid breaking the build. This difference is statistically significant (Fisher's Exact Test: $p < 0.00001$). This also fits with our survey result that developers say that using CI makes them less worried about breaking the build. One respondent (P219) added that CI *"Prevents contributors from releasing breaking builds"*. By not merging in potential problem pull requests, developers can avoid breaking their builds.

Observation

CI build status can help developers avoid breaking the build by not merging problematic pull requests into their projects.

RQ13: *Do pull requests with CI builds get accepted faster (in terms of calendar time)?*
Once a pull request is submitted, the code is not merged until the pull request is accepted. The sooner a pull request is accepted, the sooner the code is merged into the project. In the previous question, we saw that projects using CI accept fewer (i.e., reject or ignore more) pull requests than projects not using CI. In this question, we consider only accepted pull requests, and ask whether there is a difference in the time it takes for projects to accept pull requests with and without CI. One reason developers gave for using CI is that it makes integration easier. One respondent (P183) added *"To be more confident when merging PRs"*. If integration is easier, does it then translate into pull requests being integrated faster?

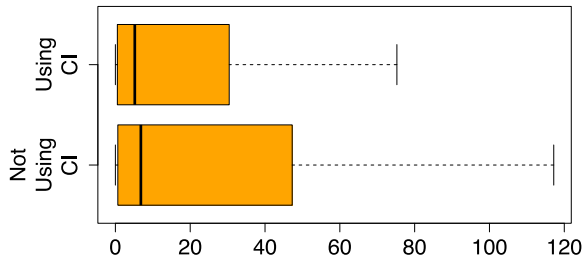

Figure 6: Distribution of time to accept pull requests, in hours

Figure 6 shows the distributions of the time to accept pull requests, with and without CI. To compute these results, we select, from our depth corpus, all the pull requests that were accepted, both with and without build information from the CI server. The mean time with CI is 81 hours, but the median is only 5.2 hours. Similarly, the mean time without CI is 140 hours, but the median is 6.8 hours. Comparing the median time to accept the pull requests, we find that the median pull request is merged 1.6 hours faster than pull requests without CI information. This difference is statistically significant (Wilcoxon, $p < .0000001$).

Observation

CI build status can make integrating pull requests faster. When using CI, the median pull request is accepted 1.6 hours sooner.

Table 9: Percentage of builds that succeed by pull request target

Pull Request Target	Percent Passed Builds
Master	72.03
Other	65.36

RQ14: *Do CI builds fail less on master than on other non-master branches?* The most popular reason that participants gave for using CI was that it helps avoid breaking the build. Thus, we analyze this claim in the depth corpus. Does the data show a difference in the way developers use CI with the master branch vs. with the other branches? Is there any difference between how many builds fail on master vs. on the

other branches? Perhaps developers take more care when writing a pull request for master than for another branch.

Table 9 shows the percentage of builds that pass in pull requests to the master branch, compared to all other branches. We found that pull requests are indeed more likely to pass when they are on master.

Observation

CI builds on the master branch pass more often than on the other branches.

5. IMPLICATIONS

We offer practical implications of our findings for researchers, developers, and tool builders.

Researchers

RQ1, RQ3, RQ4, RQ5: CI is not a "fad" but is here to stay. Because CI is widely used and more projects are adopting it, and has not yet received much attention from the research community, it is time for researchers to study its use and improve it, e.g., automate more tasks (such as setting up CI). We believe that researchers can contribute many improvements to the CI process once they understand the current state-of-the-practice in CI.

RQ2: Similarly with how GitHub has become the main gateway for researchers who study software, we believe Travis CI can become the main gateway for researchers who study CI. Travis offers a wealth of CI data, accessible via public API. Therefore, researchers can maximize their impact by studying a single system.

RQ7, RQ8: We found evidence of frequent evolution of CI configuration files (similar evolution was found for Makefiles [21]), so researchers can focus on providing support for safe automation of changes in configuration files, e.g., via safe refactoring tools.

RQ8: We confirmed that continuously running CI takes a non-trivial amount of time (and resources), so the testing research community should investigate methods for faster build and test, similar to the ongoing efforts on TAP at Google [4, 33, 34, 51, 52], Tools for Software Engineers (TSE) at Microsoft [16], or regression testing [38, 55].

RQ6, Table 4 : The most common reason why developers do not use CI is unfamiliarity with CI, so there is tremendous opportunity for providing educational resources. We call upon university educators to enrich their software engineering curriculum to cover the basic concepts and tooling for CI.

Developers

RQ3, Table 3: The data shows that CI is more widely embraced by the projects that use dynamically typed languages (e.g., 64% of 2721 Ruby projects use CI, compared with only 20% of 2745 Objective-C projects that use CI). To mitigate the lack of a static type system, developers that use dynamically typed languages should use CI to run tests and help catch errors early on.

RQ13: Our analysis of the depth corpus shows that the presence of CI makes it easier to accept contributions in open-source projects, and this was also indicated by several survey respondents, e.g., *"CI gives external contributors confidence that they are not breaking the project" (P310)*. Considering other research [43] that reports a lack of diversity in open-source projects, attracting new contributors is desirable. Thus, projects that aim to diversify their pool of contributors should consider using CI.

RQ7, RQ9: Because the average times for a single CI build is fairly short, and CI configurations are maintainable, it appears that the benefits of CI outweigh the costs. Thus, developers should use CI for their projects.

RQ3, RQ11, RQ12, RQ14: The use of CI correlates with positive outcomes, and CI has been adopted by the most successful projects on GitHub, so developers should consider CI as a best practice and should use it as widely as possible.

Tool Builders

RQ6: CI helps catching bugs, but not locating them. The CI build logs often bury an important error message among hundreds of lines of raw output. Thus, tool builders that want to improve CI can focus on new ways to integrate fault-localization techniques into CI.

RQ1, RQ7, RQ8: Despite wide adoption, there are many projects that have yet to use CI. Tool builders could parse build files [56], and then generate configuration files necessary for CI. By automating this process, tool builders can lower the entry barrier for developers who are unfamiliar with CI.

6. THREATS TO VALIDITY

6.1 Construct

Are we asking the right questions? We are interested in assessing the usage of CI in open-source projects. To do this we have focused on *what*, *how*, and *why* questions. We think that these questions have high potential to provide unique insight and value for different stakeholders: developers, tool builders, and researchers.

6.2 Internal

Is there something inherent to how we collect and analyze CI usage data that could skew the accuracy of our results?

Once a CI server is configured, it will continue to run until it is turned off. This could result in projects configuring a CI server, and then not taking into account the results as they continue to do development. However, we think this is unlikely because Travis CI and GitHub have such close integration. It would be difficult to ignore the presence of CI when there are visual cues all throughout GitHub when a project is using CI.

Some CI services are run in a way such that they cannot be detected from the information that is publicly available in the GitHub repository. This means that we could have missed some projects. However, this would mean that we are underestimating the extent to which CI has been adopted.

Despite a 9.8% response rate to our survey, still over 90% of our targeted population did not respond. We had no control over who responded to our survey, so it may suffer from self-selection bias. We think this is likely because 92% of our survey participants reported using CI, much higher than the percentage of projects we observed using CI in the data. In order to mitigate this, we made the survey short

and provided a raffle as incentive to participate, to get the most responses as possible.

6.3 External

Are our results generalizable for general CI usage? While we analyzed a large number of open-source repositories, we cannot guarantee that these results will be the same for proprietary (closed-source) software. In fact, we consider it very likely that closed-source projects would be unwilling to send their source over the internet to a CI service, so our intuition is that they would be much more likely to use a local CI solution. Further work should be done to investigate the usage of CI in closed-source projects.

Because we focused on Travis CI, it could be that other CI services are used differently. As we showed in RQ2, Travis CI was the overwhelming favorite CI service to use, so by focusing on that we think our results are representative.

Additionally, we only selected projects from GitHub. Perhaps open-source projects that have custom hosting also would be more likely to have custom CI solutions. More work is needed to determine if these results generalize.

7. RELATED WORK

We group our related work into three different areas: (i) *CI usage*, (ii) *CI technology*, and (iii) *related technology*.

CI Usage The closest work to ours is by Vasilescu et al. [53] who present two main findings. They find that projects that use CI are more effective at merging requests from core members, and the projects that use CI find significantly more bugs. However, the paper explicitly states that it is a preliminary study on only 246 GitHub projects, and treats CI usage as simply a boolean value. In contrast, this paper examines 34,544 projects, 1,529,291 builds, and 442 survey responses to provide detailed answers to 14 research questions about CI usage, costs, and benefits.

A tech report from Beller et al. [25] performs an analysis of CI builds on GitHub, specifically focusing on Java and Ruby languages. They answer several research questions about tests, including "How many tests are executed per build?", "How often do tests fail?", and "Does integration in different environments lead to different test results?". These questions however, do not serve to comprehensively support or refute the productivity claims of CI.

Two other papers [44, 46] have analyzed a couple of case studies of CI usage. These are just two case studies total, unlike this paper that analyzes a broad and diverse corpus.

Leppänen et al. [45] interviewed developers from 15 software companies about what they perceived as the benefits of CI. They found one of the perceived benefits to be more frequent releases. One of their participants said CI reduced release time from six months to two weeks. Our results confirm that projects that use CI release twice as fast as projects that do not use CI.

Beller et al. [24] find that developers report testing three times more often than they actually do test. This over-reporting shows that CI is needed to ensure tests are actually run. This confirms what one of our respondents (P287) said: *"It forces contributors to run the tests (which they might not otherwise do)"*. Kochhar et al. [42] found that larger Java open-source projects had lower test coverage rates, also suggesting that CI can be beneficial.

CI technology Some researchers have proposed approaches to improve CI servers by having servers communicate dependency information [31], generating tests during CI [30], or selecting tests based on code churn [41]. Also researchers [27] have found that integrating build information from various sources can help developers. In our survey, we found that developers do not think that CI helps them locate bugs; this problem has been also pointed out by others [28].

One of the features of CI systems is that they report the build status so that it is clear to everyone. Downs et al. [32] developed a hardware based system with devices shaped like rabbits which light up with different colors depending on the build status. These devices keep developers informed about the status of the build.

Related Technology A foundational technology for CI is build systems. Some ways researchers have tried to improve their performance has been by incremental building [35] as well as optimizing dependency retrieval [29].

Performing actions continuously can also bring extra value, so researchers have proposed several activities such as continuous test generation [54], continuous testing (continuously running regression tests in the background) [50], continuous compliance [36], and continuous data testing [47].

8. CONCLUSIONS

CI has been rising as a big success story in automated software engineering. In this paper we study the usage, the growth, and the future prospects of CI using data from three complementary sources: (i) 34,544 open-source projects from GitHub, (ii) 1,529,291 builds from the most commonly used CI system, and (iii) 442 survey respondents. Using this rich data, we investigated 14 research questions.

Our results show there are good reasons for the rise of CI. Compared to projects that do not use CI, projects that use CI: (i) release twice as often, (ii) accept pull requests faster, and (iii) have developers who are less worried about breaking the build. Therefore, it should come as no surprise that 70% of the most popular projects on GitHub heavily use CI.

The trends that we discover point to an expected growth of CI. In the future, CI will have an even greater influence than it has today. We hope that this paper provides a call to action for the research community to engage with this important field of automated software engineering.

9. ACKNOWLEDGMENTS

We thank CloudBees for sharing with us the list of open-source projects using CloudBees, Travis for fixing a bug in their API to enable us to collect all relevant build history, and Amin Alipour, Denis Bogdanas, Mihai Codoban, Alex Gyori, Kory Kraft, Nicholas Lu, Shane McKee, Nicholas Nelson, Semih Okur, August Shi, Sruti Srinivasa Ragavan, and the anonymous reviewers for their valuable comments and suggestions on an earlier version of this paper.

This work was partially funded through the NSF CCF-1421503, CCF-1439957, and CCF-1553741 grants.

10. REFERENCES

[1] 7 reasons why you should be using continuous integration. https://about.gitlab.com/2015/02/03/7-reasons-why-you-should-be-using-ci/. Accessed: 2016-04-24.

[2] AppVeyor. https://www.appveyor.com/. Accessed: 2016-04-26.

[3] The benefits of continuous integration. https://blog.codeship.com/benefits-of-continuous-integration/. Accessed: 2016-04-24.

[4] Build in the cloud. http://google-engtools.blogspot.com/2011/08/build-in-cloud-how-build-system-works.html.

[5] CircleCI. https://circleci.com/. Accessed: 2016-04-26.

[6] CloudBees. http://cloudbees.com/. Accessed: 2016-04-26.

[7] Continuous integration. https://www.thoughtworks.com/continuous-integration. Accessed: 2016-04-24.

[8] Continuous integration is dead. http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html. Accessed: 2016-04-24.

[9] CruiseControl. http://cruisecontrol.sourceforge.net/. Accessed: 2016-04-21.

[10] CrunchBase. https://www.crunchbase.com/organization/travis-ci#/entity. Accessed: 2016-04-24.

[11] Google Search Trends. https://www.google.com/trends/. Accessed: 2016-04-24.

[12] Jenkins. https://jenkins.io/. Accessed: 2016-04-21.

[13] Restkit. https://github.com/RestKit/RestKit. Accessed: 2016-04-29.

[14] Stackoverflow. http://stackoverflow.com/questions/214695/what-are-some-arguments-against-using-continuous-integration. Accessed: 2016-04-24.

[15] Team Foundation Server. https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx. Accessed: 2016-04-21.

[16] Tools for software engineers. http://research.microsoft.com/en-us/projects/tse/. Accessed: 2016-04-24.

[17] Travis CI. https://travis-ci.org/. Accessed: 2016-04-21.

[18] Werker. http://wercker.com/. Accessed: 2016-04-26.

[19] Why don't we use continuous integration? https://blog.inf.ed.ac.uk/sapm/2014/02/14/why-dont-we-use-continuous-integration/. Accessed: 2016-04-24.

[20] Yaml: Yaml ain't markup language. http://yaml.org/. Accessed: 2016-04-24.

[21] J. M. Al-Kofahi, H. V. Nguyen, A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Detecting semantic changes in Makefile build code. In *ICSM*, 2012.

[22] J. Allspaw and P. Hammond. 10+ deploys per day: Dev and ops cooperation at Flickr. https://www.youtube.com/watch?v=LdOe18KhtT4. Accessed: 2016-04-21.

[23] K. Beck. Embracing change with Extreme Programming. *Computer*, 32(10):70–77, 1999.

[24] M. Beller, G. Gousios, and A. Zaidman. How (much) do developers test? In *ICSE*, 2015.

[25] M. Beller, G. Gousios, and A. Zaidman. Oops, my tests broke the build: An analysis of travis ci builds with github. Technical report, PeerJ Preprints, 2016.

[26] G. Booch. *Object Oriented Design with Applications*. Benjamin-Cummings Publishing Co., Inc., 1991.

[27] M. Brandtner, E. Giger, and H. C. Gall. Supporting continuous integration by mashing-up software quality information. In *CSMR-WCRE*, 2014.

[28] M. Brandtner, S. C. Müller, P. Leitner, and H. C. Gall. SQA-Profiles: Rule-based activity profiles for continuous integration environments. In *SANER*, 2015.

[29] A. Celik, A. Knaust, A. Milicevic, and M. Gligoric. Build system with lazy retrieval for Java projects. In *FSE*, 2016.

[30] J. C. M. de Campos, A. Arcuri, G. Fraser, and R. F. L. M. de Abreu. Continuous test generation: Enhancing continuous integration with automated test generation. In *ASE*, 2014.

[31] S. Dösinger, R. Mordinyi, and S. Biffl. Communicating continuous integration servers for increasing effectiveness of automated testing. In *ASE*, 2012.

[32] J. Downs, B. Plimmer, and J. G. Hosking. Ambient awareness of build status in collocated software teams. In *ICSE*, 2012.

[33] S. Elbaum, G. Rothermel, and J. Penix. Techniques for improving regression testing in continuous integration development environments. In *FSE*, 2014.

[34] J. Engblom. Virtual to the (near) end: Using virtual platforms for continuous integration. In *DAC*, 2015.

[35] S. Erdweg, M. Lichter, and M. Weiel. A sound and optimal incremental build system with dynamic dependencies. In *OOPSLA*, 2015.

[36] B. Fitzgerald, K. J. Stol, R. O'Sullivan, and D. O'Brien. Scaling agile methods to regulated environments: An industry case study. In *ICSE*, 2013.

[37] M. Fowler. Continuous Integration. http://martinfowler.com/articles/originalContinuousIntegration.html. Accessed: 2016-04-21.

[38] M. Gligoric, L. Eloussi, and D. Marinov. Practical regression test selection with dynamic file dependencies. In *ISSTA*, 2016.

[39] G. Gousios. The GHTorrent dataset and tool suite. In *MSR*, 2013.

[40] J. Humble. Evidence and case studies. http://continuousdelivery.com/evidence-case-studies/. Accessed: 2016-04-29.

[41] E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, and M. Castell. Supporting continuous integration by code-churn based test selection. In *RCoSE*, 2015.

[42] P. S. Kochhar, F. Thung, D. Lo, and J. L. Lawall. An empirical study on the adequacy of testing in open source projects. In *APSEC*, 2014.

[43] V. Kuechler, C. Gilbertson, and C. Jensen. Gender differences in early free and open source software joining process. In *IFIP*, 2012.

[44] E. Laukkanen, M. Paasivaara, and T. Arvonen. Stakeholder perceptions of the adoption of continuous integration: A case study. In *AGILE*, 2015.

[45] M. Leppänen, S. Mäkinen, M. Pagels, V. P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistä. The highways and country roads to continuous deployment. *IEEE Software*, 2015.

[46] A. Miller. A hundred days of continuous integration. In *AGILE*, 2008.

[47] K. Muşlu, Y. Brun, and A. Meliou. Data debugging with continuous testing. In *FSE*, 2013.

[48] V. One. 10th annual state of Agile development survey. https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf, 2016.

[49] Puppet and DevOps Research and Assessments (DORA). 2016 state of DevOps Report. https://puppet.com/system/files/2016-06/2016%20State%20of%20DevOps%20Report_0.pdf, 2016.

[50] D. Saff and M. D. Ernst. Continuous testing in Eclipse. In *ICSE*, 2005.

[51] Testing at the speed and scale of Google, Jun 2011. http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html.

[52] Tools for continuous integration at Google scale, October 2011. http://www.youtube.com/watch?v=b52aXZ2yi08.

[53] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in GitHub. In *FSE*, 2015.

[54] Z. Xu, M. B. Cohen, W. Motycka, and G. Rothermel. Continuous test suite augmentation in software product lines. In *SPLC*, 2013.

[55] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *STVR*, 22(2):67–120, 2012.

[56] S. Zhou, J. M. Al-Kofahi, T. N. Nguyen, C. Kästner, and S. Nadi. Extracting configuration knowledge from build files with symbolic analysis. In *RELENG*, 2015.