Continuous Integration (CI) Needs and Wishes for Developers of Proprietary Code

Michael Hilton, Nicholas Nelson, Danny Dig
School of EECS, Oregon State University
{hiltonm,nelsonni,digd}@oregonstate.edu

Timothy Tunnell, Darko Marinov
CS Department, University of Illinois
{tunnell2,marinov}@illinois.edu

***Abstract*—Continuous integration (CI) systems automate the compilation, building, and testing of software. Despite CI being one of the most widely used processes in software engineering, we do not know what motivates developers to use CI, and what barriers and unmet needs they face. Without such knowledge developers make easily avoidable errors, managers reduce the productivity of developers by making misinformed decisions, tool builders invest in the wrong direction, and researchers miss many opportunities for improving the software engineering practice. Given the large fraction of proprietary code development, understanding how proprietary developers are using CI is vital to improving it.**

We present the first study of how CI is used in the proprietary development of software. We conduct 16 semi-structured interviews with developers from different industries and development scale. We generalize these findings by surveying 523 developers from all over the world. We find that 78% of developers feel more productive when using CI, and 85% think that using CI causes developers to give more value to automated testing. However, unlike open-source developers, many proprietary developers want to use CI but are not allowed to do so. 50% of developers have problems troubleshooting CI builds, and more than half of developers want easier configurations for CI tools and services.

I. Introduction

Continuous integration (CI) systems automate the compilation, building, and testing of software. Adopting CI usually has positive effects on software development. For example, the "State of DevOps" report [1], a survey of over 4,600 technical professionals from around the world, finds CI to be an indicator of "high performing IT organizations". Researchers [2] report that adopting CI allowed a product group at HP to reduce development costs by 78%.

CI Usage is widespread throughout the software development industry. The "State of Agile" industry survey [3], with 3,880 participants, found 50% of respondents use CI. In our previous work [4], we found that 40% of the 34,000 most popular open-source projects on GitHub use CI, and the most popular projects are more likely to use CI (70% of the top 500 projects).

Despite the widespread adoption of CI, there are still many unanswered questions about CI. In one study, Vasilescu et al. [5] show that CI correlates with positive quality outcomes. In our previous work [4], we provide answers to questions about the usage, costs, and benefits of CI. However, both of these studies were conducted on open-source projects, and do not answer any questions about the usage of CI by projects with proprietary code.

Open-source projects are often considered fundamentally different from proprietary-code projects [6]. Differences between open-source and proprietary-code projects include that open-source projects foster more creativity [7], have fewer defects [7], and assign work differently [8]. However, similarities have also been found, including how GitHub is used [9], overall project complexity [7], and amount of modularity [7].

There are still many gaps in our knowledge about why developers use CI, and what barriers they face, especially for projects of proprietary code. What motivates developers to use CI? What are the barriers that developers face when using CI? What needs do developers have that are unmet by their current CI system(s)? Do proprietary developers have the same answers to these questions as open-source developers?

Without answers to these questions, *developers* cannot take advantage of the benefits of CI, which may lead them to be less productive, write code and tests of lower quality, and spend more time worrying about their builds breaking. *Managers* can refuse to implement CI thinking it is a passing fad, and potentially miss out on the benefits it could bring to their project. Likewise, *tool builders* can waste time and effort on tools that will not meet their customers' needs, while failing to provide needed enhancements, such as easier CI configuration. *Researchers* have a blind spot which prevents them from providing solutions to the hard problems that practitioners face, such as how to parallelize tests properly, or better ways to do fault localization when the CI builds break.

To answer these questions, we employ established research methodologies from prior software engineering research [14]–[18]. The primary methodology we employ in this work is interviews with 16 software developers from 14 different companies. To the best of our knowledge, we are the first to interview developers specifically about CI. To quantify and extend our findings, we deployed a survey that had 523 participants, of which 95% are from industry, and 70% percent have 7 or more years of experience. The interviews provide depth to our results, while the survey provides breadth. With this data, we answer the following four research questions:

RQ1: *Why do developers use CI?*
RQ2: *What benefits do developers experience when using CI?*
RQ3: *What barriers do developers face when using CI?*
RQ4: *What unmet needs do developers have with CI tools?*

We find that 70% of developers use CI because it helps them catch errors earlier, lets them worry less about breaking

the build, and provides a common build environment. We also find that 78% of developers think that CI helps them be more productive, and 85% believe that using CI causes developers to give more value to automated testing. We find that unlike open-source developers, many proprietary-code developers would like to use CI, but are unable to do so, often because they are not allowed to by their management. When using CI, the most common difficulties experienced by developers are troubleshooting a build, overly long build times, and difficulties automating the build process.

We examine gaps in current CI solutions. We find that the most common way developers say that CI solutions are not meeting their needs include that CI tools are difficult to configure, have poor integration with other tools, and do not provide sufficient debugging assistance.

Finally, we present implications for four audiences. For example, managers should encourage developers to use CI. Tool builders should focus on making it easier to configure CI tools. Developers can use CI to unlock the value of automated testing. Researchers have a lot to bring to the CI community, such as helping with fault localization and test parallelization.

This paper makes the following contributions:

1) We conduct exploratory semi-structured interviews with 16 developers, then quantify these findings with a survey of 523 developers.
2) We provide an empirically justified set of developers' motivation for using CI.
3) We expose gaps between developers' needs and existing tooling for CI.
4) We present actionable implications that developers, managers, tool builders, and researchers can build on.

The interview script along with the survey questions and responses can be found on this study's website: **HTTP://COPE.EECS.OREGONSTATE.EDU/ICSE17**

II. Background

A. History

One of the rules [10] that Extreme Programming proposed was "Integrate Often". This rule states that changes should be merged in every few hours, and at least once a day. A few years later, iIn 2000, Martin Fowler [11] authored a blog post which presented CI as it is currently understood. The core premise of CI is that the more often a project integrates, the better off it is. Automation is the key to making this possible , according to Fowler. The entire build process should be automated, including retrieving the sources, compiling, linking, and running automated tests. The system should output "pass" or "fail" to indicate whether the build was successful. The automated build process can be started manually, or triggered automatically by actions such as checking new code into a version-control system (VCS).

Fowler also developed the first CI system, CruiseControl [12], which was released in 2001. Today there are over 40 CI systems available. Some of the best known systems include Jenkins [13] (previously called Hudson), Travis CI [14], and Microsoft Team Foundation Server (TFS) [15]. The first systems usually ran locally, as Jenkins and TFS still do. However, CI as a service, e.g., Travis CI, has become increasingly more popular recently.

B. CI concepts

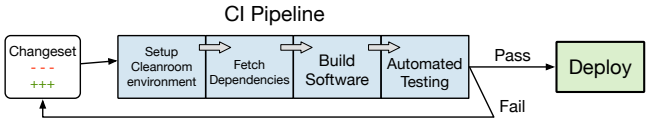

Fig. 1. CI Pipeline: A completed *changeset triggers* the pipeline. The result of the pipeline is a "pass" or "fail" result of the integration.

We asked our interview participants to describe their CI pipeline. While not all pipelines are the same, they generally share some common elements. We now present some of the core elements of a CI pipeline.

Changesets are a group of changes that a developer makes to the code. They could be a single commit, or a group of commits, but they should be a complete change, so that after the changeset is applied, it should not break the program.

When a CI system or service observes a change made by developers, this *triggers* a CI event. How and when the CI is triggered is based on how the CI is configured. The most common way to trigger CI is when a commit is pushed to a repository (e.g. master branch in Git).

For the CI to verify the code, it is important for this to happen in a *cleanroom* environment. The concept of cleanroom in CI is that the evaluation of the changeset is done in a clean environment. The automated build script should be able to start with a clean environment, and build the product from scratch. Many developers are using containers to implement cleanroom builds. Some of the more popular container technologies include Docker [16] and Vagrant [17].

An important step in the CI pipeline is verifying that the changeset was integrated correctly into the application. One common method is a regression test suite, including unit tests and integration tests. The CI system can also run a variety of other tests, for example, linting, or evaluating tests coverage.

The last step is to deploy the built and verified artifact. We found some developers consider deploying to be part of CI, and others consider it to be a separate process, *Continuous Deployment* (CD).

III. Methodology

To accomplish this, we employ established research methodologies from prior software engineering research [18]–[22]. The primary methodology we employ in this work is interviews with software developers. We then confirm our results by running a large-scale survey.

Interviews are a qualitative method and are effective at discovering the knowledge and experiences of the participants. However, they often have a limited sample size [22]. Surveys are a quantitative technique that summarizes information over a larger sample size and thus provides increased generalizability [22]. Together, they provide a much clearer picture than

either can provide alone. The interviews provide a depth of information, while the surveys provide breadth.

We first use interviews to elicit developers experiences and expectations when working with CI, and we build a taxonomy of motivations, benefits, problems, strategies, and unmet needs. We then deploy a survey to quantify and refine this taxonomy.

The interview script along with the survey questions and responses can be found on this study's website [23]

A. Interviews

We used semi-structured interviews [22] "which include a mixture of open-ended and specific questions, designed to elicit not only the information foreseen, but also unexpected types of information."

We interviewed 16 developers from 14 different companies, including large software companies, CI service companies, small development companies, a telecommunications company, and software consultants. Our participants had an average of over seven years of development experience. They all used CI, often a variety of CI tools, including Jenkins [13], TravisCI [14], Wercker [24], CruiseControl.NET [25], CircleCI [26], TeamCity [27], XCode Bots [28], Buildbot [29], appVeyor [30], as well as proprietary CI solutions. Each interview lasted between 30 and 60 minutes, and the participants were offered a US$50 Amazon gift card at the end of the interview.

The interviews were based on the research questions presented in Section I. The following is an example of some of the questions we asked in the interview:

- How would you define Continuous Integration (CI)?
- Tell me about the last time you used CI.
- What tasks prompt you to interact with your current CI tools?
- Comparing projects that do use CI with those that don't, what differences have you observed?
- What, if anything, would you like to change about your current CI solution?

We coded the interviews using established guidelines from the literature [31] and followed the guidance from Campbell et al. [32] on specific issues related to coding semi-structured interview data, such as segmentation, codebook evolution, and achieving coder agreement.

The first author segmented the transcript from each interview by units of meaning [32]. The first two authors then started collaborating on coding the segmented interviews, using the negotiated agreement technique to achieve agreement [32]. After the third interview, the authors created a new codebook to replace their initial codes, and then re-coded the first three interviews with the new codebook. This codebook was then used for the rest of the coding, resolving differences through the addition, deletion, merger, or better descriptions of the codes. It took eight interviews to stabilize the codes. After the eighth interview, the first and second author independently coded the remaining interviews. Each interview was then reviewed by both authors, and agreement was reached using the negotiated agreement technique. Because agreement is negotiated, there is no need to measure inter-rater agreement. Our final codebook contained 25 codes divided into 4 groups: demographics, systems/tools, process, and human CI interaction. The full codeset is available on our companion site [23].

B. Surveys

We created a survey with 21 questions to quantify the findings from our semi-structured interviews. To compare results between open-source and proprietary developers, we included two questions from our previous work [4], where we focused on open-source development. We derived the rest of the questions directly from the interview results. The survey consisted of multiple choice questions, with a final open-ended text field to allow participants to share any additional information about CI. We ensure completeness by including an "other" field where appropriate. We prevent the order of answers from biasing our participants by using a survey tool which randomized the order of answers for the survey participants. To maximize participation, we followed guidelines from the literature [33], including keeping the survey as short as possible, and raffling a gift certificate (specifically, a $50 USD Amazon gift card) to survey participants. We recruited participants by advertising our survey on social media[1]. We collected 523 complete responses, and a total of 691 survey responses.

Over 50% of our participants had over 10 years experience, and over 80% had 4 or more years experience. 93% of our developers write code for proprietary projects. In contrast with our previous open-source study [4], which found that 90% of open-source CI usage was Travis CI, we found that 75% of the developers from this survey were using Jenkins.

IV. Motivations

TABLE I
DEVELOPERS' MOTIVATION FOR USING CI. N=500

Reason	total (percent)
CI helps us catch errors earlier	375 (75%)
CI makes us less worried about breaking our builds	359 (72%)
CI provides a common build environment	349 (70%)
CI helps us deploy more often	339 (68%)
CI allows faster iterations	284 (57%)
CI makes integration easier	283 (57%)
CI can enforce a specific workflow	198 (40%)
CI allows testing across multiple platforms	143 (29%)
CI lets us spend less time debugging	121 (24%)
CI allows running tests on more powerful hardware	109 (22%)

In this section we present the reasons that motivate developers to use CI. We now present each motivational benefit ordered by how popular it was among our survey participants. Table I shows each reason, and the number of developers who chose each reason. A total of 500 survey participants answered this question.

CI helps catch bug earlier Preventing the deployment of broken code is a major concern for developers. Finding and fixing bugs in production can be an expensive and stressful endeavor. Kerzazi and Adams [34] reported that 50% of all post-release failures were because of bugs. We would expect

[1]Facebook, Twitter, and reddit/r/SampleSize

that preventing bugs from going into production is a major concern for developers. Indeed, many interview participants said that one of the biggest benefits of CI was that it identifies bugs early on, keeping them out of the production code. For example, P3 said:

> *[CI] does have a pretty big impact on [catching bugs]. It allows us to find issues even before they get into our main repo, ... rather than letting bugs go unnoticed, for months, and letting users catch them.*

Less worry about breaking the build. Kerzazi et al. [35] reported that for one project, up to 2,300 man-hours were lost over a six month period due to broken builds. Not surprisingly, this was a common theme among interview participants. For instance, P3 discussed how often this happened before CI:

> *...and since we didn't have CI it was a nightmare. We usually tried to synchronize our changes, ... [but] our build used to break two or three times a day.*

P2 talked about the repercussions of breaking the build:

> *[When the build breaks], you gotta wait for whoever broke it to fix it. Sometimes they don't know how, sometimes they left for the day, sometimes they have gone on vacation for a week. There were a lot of points at which all of us, a whole chunk of the dev team was no longer able to be productive.*

Providing a common build environment. One challenge developers face is ensuring the environment contains all dependencies needed to build the software. By starting the CI process with a clean environment, fetching all the dependencies, and then building the code each time, developers can be assured that they can always build their code. Several developers told us that if the code doesn't build on the CI, then the build is broken, regardless of how it behaves on an individual developer's machine. For example, P5 said:

> *...If it doesn't work here (on the CI), it doesn't matter if it works on your machine.*

CI helps projects deploy more often. In our previous work [4] we found that open-source projects that use CI deploy twice as often as projects that do not use CI. In our interviews, some developers told us that they felt that CI helped them deploy more often. However, other developers told us that CI enabled them to have shorter development cycles than they otherwise would have, even if they did not deploy often because of business reasons. For example, P14 said:

> *[Every two weeks] we merge into master, and consider that releasable. We don't often release every sprint, because our customer doesn't want to. Since we are services company, not a products company, its up to our customer to decide if they want to release, but we ensure every two weeks our code is releasable if the customer chooses to do so.*

CI allows faster iterations. Participants told us that running CI for every change allows them to quickly identify when the current change set will break the build, or will cause problems in some other location of the codebase. Having this immediate feedback enables much faster development cycles. This speed allows developers to make large changes quickly, without introducing a large amount of bugs into the codebase. P15 stated:

> *We were able to run through up to 10 or 15 cycles a day, running through different tests, to find where we were, what solutions needed to be where. Without being able to do that, without that speed, and that feedback, there is no way we could have accomplished releasing the software in the time frame required with the quality we wanted.*

CI makes integration easier. The initial blog post [11] that introduced the concept of CI, presented it as a way to avoid painful integrations. However, while developers do think CI makes integration easier, it is not the primary reason that motivates developers to use CI. During our interviews, several participants contrasted CI with SVN, the most common version-control system prior to Git. For many developers, they see their VCS as the solution to difficult integrations, not the CI. It is also interesting to note that while easier integration is the original purpose of CI, developers have found more benefit from CI "side effects."

Enforcing a specific workflow Prior to CI, there was no common way for tools to enforce a specific workflow (e.g., ensuring all tests are run before accepting changes). This is especially a concern for distributed and open-source teams, where it is harder to overcome tooling gaps through informal communication channels. However, with CI, not only are all the tests run on every changeset, but everyone knows what the results are. Everyone on the team is aware when code breaks the tests or the builds, without having to download the code, and verify the test results on their own machine. This can help find bugs faster and increase team awareness, both of which are important parts of code review [36] P16 told us that he was pretty sure that contributors to his project were not running the tests routinely, before they added CI to their project.

Test across all platforms. CI allows a system to be tested on all major platforms (Windows, Linux, and OS X), without each environment being setup locally by each developer. For example, P16 stated:

> *We are testing across more platforms now, it is not just OS X and Linux, which is mostly what developers on projects run. That has been useful. You wouldn't think that there is much difference between Chrome on Windows and OS X and such, but you get these weird fringe cases occasionally that you need to fix, so [CI] has made [our software] more robust for multiple platforms.*

Interestingly, only 143 (29%) of survey participants listed this as a motive for using CI. In fact, one survey participant responded to our open-ended question at the end of the survey:

> *Simplifying CI across platforms could be easier. We currently want to test for OS X, Linux and Windows and need to have 3 CI services.*

While this is a benefit already realized for some participants,

others see this as an area in with CI could still show substantial improvement.

Observation

Developers extol the benefits of CI, but many of the benefits they cite come from running automated tests more often.

V. Experiences

We asked interview participants about their experiences when using CI. We then quantify these findings by asking our survey participants which benefits they had experienced as well. Because not every survey participant has enough experience to answer every questions, we allowed participants to check a "do not know" option, which we do not count in the results.

Projects with CI have higher quality tests Interview participants told us that because projects that use CI run their automated tests more often, and the results are visible to the entire team, this motivates developers to have higher quality tests. To confirm this, we asked the same question of survey participants. Figure 2 shows that the survey participants overwhelming agree (75%) that projects with CI have higher quality tests.

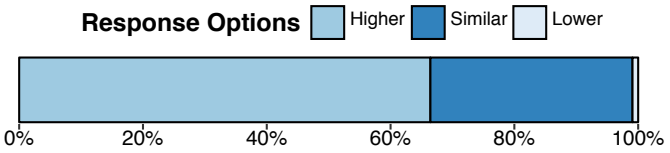

Fig. 2. Do projects with CI have (higher/similar/lower) test quality? N=444

TABLE II
Code coverage improvement, by quartiles

Quartile	Num Projects	Start Avg Coverage %	End Avg Coverage %	Avg Improvement
1st	183	46.9	60.1	13.2
2nd	184	76.3	81.6	5.3
3rd	184	89.8	91.1	1.3
4th	184	97.8	94.9	-2.9

In our interviews, participants told us that using CI had been a part of increasing test coverage, resulting in higher quality tests. P8 said:

> *...but it was enough to start shifting the culture within my company. And we jumped the coverage from a single digit to half of the code base in one year.*

Based on these claims, we decided to conduct a new experiment with the data from our previous paper [4] that had studied the interaction of GitHub projects and Travis CI service (but not any other service). We examined the 4335 most popular GitHub projects by stars from our breadth corpus and found that 735 projects had usable coverage information available on the coveralls.io service [37]. We collected all the coverage information available for these projects. We could not directly measure an impact of CI, e.g., by comparing coverage information from projects that use and do not use CI, because coveralls is dependent on CI and has no data available for projects that do not use CI. Instead, we looked at the change of coverage over time for the projects that do use CI. We found is the change of coverage was largely dependent on the starting state of the projects. We sorted the projects by their starting test coverage percentage and divided them into four quartiles. We then compared the starting and ending code coverage for each project. Table II shows the result. For projects that already had a high starting test coverage (90% and greater), there was very little growth of the test coverage. However, for projects with a low starting test coverage, there were substantial improvements. This finding seems to corroborate the experiences related by participants, However, more research is needed.

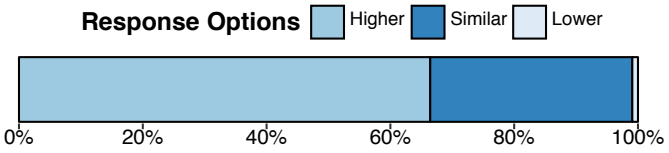

Fig. 3. Do projects with CI have (higher/similar/lower) code quality? N=438

Projects that use CI have higher code quality By writing a good automated test suite, and running it after every change, developers can quickly identify when they make a change that does not behave as anticipated, or breaks some other part of the code. P10 said:

> *CI for me is a very intimate part of my development process. ... I lean on it for confidence in all areas. Essentially, if I don't have some way of measuring my test coverage, my confidence is low. ... If I don't have at least one end-to-end test, to make sure it runs as humans expect it to run, my confidence is low. So I lean pretty heavily on CI, I use it all day everyday.*

When we asked our survey participants, they also agreed: 67% reported that they had experienced higher code quality on projects with CI.

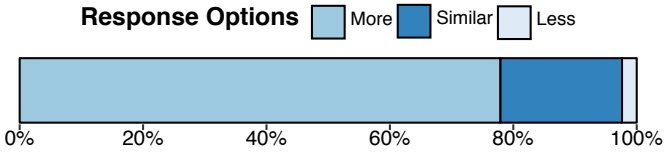

Fig. 4. Are developers on projects with CI (more/similar/less) productive? N=462

Developers on projects with CI are more productive According to the developers, CI allows developers to focus more on being productive, and to let the CI take care of boring, repetitive steps, which can be handled by automation. P2 said:

> *it just gets so unwieldy, and trying to keep track of all those bits and pieces that are moving around, ... [CI makes it] easier it is for them to just focus on what they need to do.*

Another reason interview participants gave for being more productive with CI was that CI allows for faster iterations, which helps developers be more productive. P16 said:

> *I think [CI] has made it easier to iterate more quickly, because you can have more trust that you are not breaking all the things.*

Our survey participants also felt that CI enabled developers to be more productive: 78% supported this.

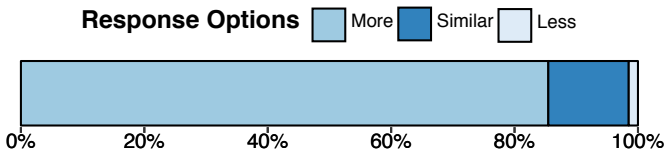

Fig. 5. Do developers on projects with CI give (more/similar/less) value to automated tests? N=469

CI causes developers to give more value to automated testing Several participants told us that before using CI, their team would write automated tests, but without seeing the CI running them, the development team felt that they were forced to write automated tests without seeing them as valuable. P11 related:

> *situations I have been in, there is no CI, but there is a test suite, and there is a vague expectation that someone is running this test sometimes. And if you are the poor schmuck that actually cares about tests, and you are trying to run them, and you can't get anything to pass, and you don't know why, and you are hunting around like "does anyone else actually do this?"*

However, due to the introduction of CI, developers were able to see their tests being run for every change set, and the whole team becomes aware when the automated tests catch an error that otherwise would have made it into the product. P16 summarized this feeling:

> *[CI] increases the value of tests, and makes us more likely to write tests, to always have that check in there. [Without CI, developers] are not always going to run the tests locally, or you might not have the time to, if it is a larger suite.*

85% survey participants agreed that having CI led developers to value their tests more.

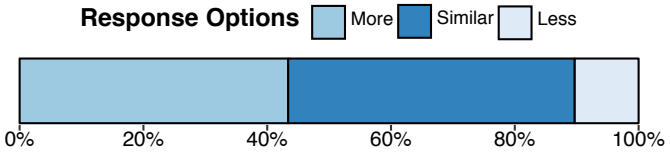

Fig. 6. CI supports similar exploratory development N=381

CI allows similar exploratory development to projects without CI When using CI, developers can more quickly receive feedback from the system about their change. This feedback can allow developers to try out a potential solution, without having to make a large commitment. P15 explained it this way:

> *developers will say, I think this is the type of change I want to do, but I don't know that it is, so we can run it through and see" and then they are able to know in quick fashion. Before they wouldn't want to do that, because if it took two months for it to get tested, they don't want to waste two months on this, but if they can get a response quickly, within a day, then they are able to do a little of experimentation.*

However, the majority of survey participants did not agree that CI leads to more exploratory development: 47% did not think so, while 43% did think so. This results leads us to conclude that more research is needed to examine the relationship between CI and exploratory development.

Observation

CI causes developers to care about automated testing. They invest more effort, which in turn leads to higher quality tests and code, and developers feeling more productive.

VI. Barriers

TABLE III
Reasons developers do not use CI. N=103

Reason	total (percent)
I don't currently use CI, but I would like to in the future	51 (49%)
The developers on my project(s) are not familiar with CI	42 (40%)
My project(s) don't have enough automated tests	38 (37%)
Automating builds for my project(s) is not worth it	18 (17%)
Setup costs are too high for CI systems	13 (12%)
CI doesn't bring value because we already do enough testing	5 (5%)
Maintenance costs are too high for CI systems	4 (4%)

A. Barriers to adopting CI

For this work, we only interviewed participants who use CI. However, for our survey, we expected our responses to also include developers who do not use CI. In our previous work [4] we asked developers why they do not use CI. However, that study specifically targeted open-source developers by only sending the survey to developers who were part of an open-source project on GitHub. To compare the responses between different target populations, we repeat this question.

The results between the two surveys were similar, with one notable exception. For open-source developers, the fourth most popular option, chosen by only 25% of participants, indicated that they were not using CI but would like to in the future. For proprietary developers, this was the most commonly chosen reason, with 51% choosing that they hope to use it in the future. One explanation for this difference could be that in open-source development, if any developer wants to add CI, they can take it upon themselves to do so. In fact, several of our interview participants indicated that if they participate in an open-source project that does not have CI, they will add it to the project. Proprietary developers, however, do not always have that freedom. In our survey, the freeform answers make it clear that many proprietary developers wish to use CI

but are not allowed to. This lack of permission could come from the company ("Company disallows"), the management ("Philosophy of managers goes against CI, but would love to use it"), or even the customer ("The customer owns the code and has not seen the value. We do not have the rights to put the code on our servers.").

Observation

Developers who wish to adopt technology solutions that will help them be more productive are often stymied by institutional hurdles.

B. Barriers when using CI

We collected a list of the problems with CI that our interview participants reported experiencing. We asked our survey participants to select up to three problems that they had experienced. If they had experienced more than three, we asked them to chose the three most common. Table IV shows the answers from the survey. There were 529 total answers to this question. We observe a fairly uniform distribution across the answers. This leads us to conclude that these problems have all been experienced by a wide cross-section of developers.

TABLE IV
PROBLEMS DEVELOPERS ENCOUNTER WHEN USING CI. N=529

Reason	total (percent)
Troubleshooting a CI build failure	266 (50%)
Overly long build times	203 (38%)
Automating the build process	176 (34%)
Lack of support for the desired workflow	163 (31%)
Setting up a CI server or service	145 (27%)
Maintaining a CI server or service	145 (27%)
Lack of tool integration	136 (26%)
Security and access controls	110 (21%)

Difficulty troubleshooting a CI build failure When a CI build fails, the developers begin the process of identifying why the build failed. Sometimes, this can be fairly straightforward. However, in some situations, this can be quite difficult. P11 described one such situation:

> *We had these builds failing, and the end result appeared to be correct, like the site was still running. So there was some sort of disjoint there. We eventually figured it out. It was a bizarre corner case that had somehow arisen.*

One way tool makers have tried to help developers is via better logging and storing test artifacts to make it easier to examine failures. One participant described how they use Sauce Labs [38], a service for automated testing of web pages, in conjunction with their CI. When a test fails on Sauce Labs, there is a recording of it that the developers can then watch to determine exactly how their test failed. Another participant described how Wercker [24] saves a container from each CI run, so one can download the container and run the code in the container to debug a failed test.

Overly long build times Because CI must verify that the current changeset is integrated correctly, it must build the code and run automated testing. This is a blocking step for developers, because they do not want to accept the changeset if they do not know if it will break anything. If this blocking step becomes too long, it can prevent developers from being effective. Almost all of our interview participants reported that their build times slowly grow over time. For example, P10 said:

> *Absolutely [our build times grow over time]. Worst case scenario it creeps with added dependencies, and added sloppy tests, and too much I/O. That's the worse case scenario for me, when it is a slow creep.*

Other participants told us they had seen build times increase because of bugs in their build tools, problems with caching and dependencies during the build process, and adding different style of tests (e.g., acceptance tests).

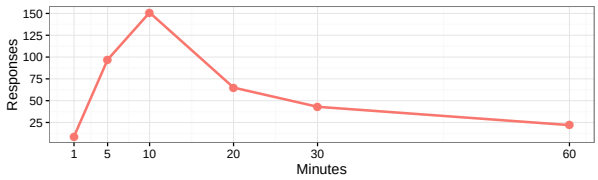

Fig. 7. Maximum acceptable build time (minutes)

Because build times are highly dependent on the project, we examined in-depth what developers meant by overly long build times. P9 said:

> *My favorite way of thinking about build times is basically, you have tea time, lunch time, or bedtime. Your builds should run in like, 5ish minutes, however long it takes to go get a cup of coffee, or in 40 minutes to 1.5 hours, however long it takes to go get lunch, or in 8ish hours, however long it takes to go and come back the next day.*

In his original blog post [11], Martin Fowler suggests most projects should try to follow the XP guideline of a ten minute build. Interestingly, when we asked our survey participants what the maximum acceptable time for a CI build to take, the answer was also 10 minutes, as shown in Figure 7.

Many of our interview participants reported having spent time and effort reducing the build time for their CI process. For example, P15 said:

> *[When the build takes too long to run], we start to evaluate the tests, and what do we need to do to speed up the environment to run through more tests in the given amount of time. ... Mostly I feel that CI isn't very useful if it takes to long to get the feedback.*

When we asked our survey participants, 78% (349/448) said they had actively worked to reduce their build times. This shows that long build times is a common barrier faced by developers who use CI.

Lack of support for the desired workflow Interview participants told us that CI tools are often designed with a specific workflow in mind. When using a tool to implement a CI process, it can be difficult to use if one is trying to use a different workflow than the one for which the tool was designed. For example, when asked how easy it is to use CI tools, P2 said:

Umm, I guess it really depends on how well you adopt their workflow. For me that's been the most obvious thing. ... As soon as you want to adopt a slightly different branching strategy or whatever else, it's a complete nightmare.

Setting up a CI server or service For our interview participants, setting up a CI server was not a concern when writing open-source code, as they can easily use one of several CI services available for free to open-source projects. We also found that large commercial projects, while very complex, often have the resources to hire dedicated people to manage the CI pipeline. However, developers on small proprietary projects do not have the resources to afford to pay for CI as a service, nor do they have the hardware and expertise needed to setup CI locally. P9, who develops an app available on the App Store, said:

[Setup] took too much time. All these tools are oriented to server setups, so I think it's very natural if you are running them on a server, but it's not so natural if you are running them on your personal computer. Setting up a new user, setting up permission for this user, that makes a lot of friction if you want to set [CI] up on your laptop.

Additionally, in the comments section of our survey, we received several comments on this issue, for example:

[We need] CI for small scale individual developers! We need better options IMO.

More research in needed to find ways for small projects to take advantage of CI.

Lack of tool integration The CI pipeline can include various different tools and applications. Often the CI pipeline includes a CI tool, an automated build tool, testing frameworks, shell scripts, containers, and more. With so many technologies in play, there are many opportunities for small issues to cause interoperability problems in the CI pipeline. Several survey participants reported specific problems, e.g., node.js and Jenkins.

Security and access controls Because CI pipelines have access to the entire source code of a given project, security and access controls are vitally important. For CI pipelines that exist entirely inside of a companies firewall, this may not be as much of a concern, but for projects using CI as a service, this can be a major issue. For developers working on company driven open-source projects, this can also be a concern. P9 said:

depending on your project, you may have an open-source project, but secrets living on or near your CI system.

Configuring the security and access controls is vital to protecting those secrets.

Observation

Despite developers' positive feelings about CI, many have encountered problems. The uniform distribution of problems suggests that there are still many areas of improvement for CI tools and processes.

TABLE V
DEVELOPER NEEDS UNMET BY CI. N=515

Reason	total (percent)
Easier configuration of CI servers or services	267 (52%)
Better tool integration	198 (38%)
Better container/virtualization support	191 (37%)
Debugging assistance	153 (30%)
User interfaces for modifying CI configurations	147 (29%)
Better notifications from CI servers or services	111 (22%)
Better security and access controls	85 (16%)
More platform support	58 (11%)

VII. NEEDS

In addition to describing problems that they had already encountered, our interview participants also described gaps where CI was not meeting their needs. From 515 answers to this question, we collected a list of these missing features, shown in Table V.

Easier configuration of CI servers or services While many CI tools offer a great deal of flexibility in how they can be used, this flexibility can require a large amount of configuration even for a simple workflow. From our interviews, we find that developers for large software companies rely on the CI engineers to ensure that the configuration is correct, and to help instantiate new configurations. Open-source developers often use CI as a service, which allows for a much simpler configuration. However, for developers trying to configure their own CI server, this can be a substantial hurdle. P8, who was running his own CI server, said:

The configuration and setup is costly, in time and effort, and yeah, there is a learning curve, on how to setup Jenkins, and setup the permissions, and the signing of certificates, and all these things. At first, when I didn't know all these tools, I would have to sort them out, and at the start, you just don't know...

It is not surprising that this is the most commonly chosen answer, especially when developers previously said that configuring a CI server is the hardest part of learning CI.

Better tool integration Our interview participants told us that they would like their CI system to better integrate with other tools. P3 remarked:

It would also be cool if the CI ran more analysis on the code, rather than just the tests. Stuff like Lint, FindBugs, or it could run bug detection tools. There are probably CIs that already do that, but ours doesn't.

It is important to note that P3 was not tasked with maintaining the CI for his project. P11 reported an experience where one individual was the gate-keeper of the CI server and would not let other developers touch it. This led to CI being rarely used by the team. The solution for this team was to switch to CI as a service, where every GitHub admin could configure their own project, which was much more successful. In our survey responses, participants added in the other field both technical problems, such as poor interoperability between node.js and

Jenkins, as well as non-technical problems, such as "The server team will not install a CI tool for us".

User interfaces for modifying CI configurations Many participants described administering their CI tools via configuration scripts. While this does work, participants expressed a desire to make these config files editable via a user interface, which they felt would be easier. P3 said:

> *Most of the stuff we are configuring could go in a UI. ... We are not modifying heavy logic. We just go in a script and modify some values. ... So all of the tedious stuff you modify by hand could go into a UI.*

In addition to 147 (29%) choosing this option in the survey, multiple participants also added "Bad UI" as a freeform answer to the question about problems experienced with CI.

Debugging assistance When asked about how to debug test failures detected by their CI, most of our participants told us that they get the console output and start their search there. These output logs can be quite large in size though, with hundreds of thousands of lines of output, from thousands of tests. This can create quite a challenge when trying to find a specific failure. P7 suggested that they would like their CI server to diff the output from the previous run and hide all the output which remained unchanged between runs. P15, who worked for a large company, had developed an in-house tool to do exactly this, to filter the output and help developers find errors faster. Additionally, among our survey participants, one of the least popular reasons for using CI was that "CI lets us spend less time debugging".

Better container/virtualization support One core concept of CI is that each build should be done in a clean environment, i.e., it should not depend on the environment containing the output from any previous builds. Participants told us that this was very difficult to achieve before software based container platforms, such as Docker or Vagrant. However, there are still times when the build fails, and in doing so, breaks the CI server. P15 explained:

> *...there will be [CI] failures, where we have to go through and manually clean up the environment.*

P3 had experienced the same issues and had resorted to building Docker containers inside other Docker containers to ensure that everything was cleaned up properly.

Better notifications from CI servers or services Almost all participants had the ability to setup notifications from their CI server, but very few found them to be useful. When asked about notifications from his CI, P7 said that he will end of with up to 20 emails from a single pull request, which he will immediately delete. Other participants did in fact find the notifications useful, though, including P10 who reads through them every morning, to refresh his memory of where he left off the day before.

Better security and access controls For CI systems that are completely located inside a company firewall, security can be handled by the outward facing firewall. However, for any external projet, maintaining proper access controls can be a real concern. P16, who uses CI as a service, described how their project uses a secure environment variable (SEV) to authenticate a browser-based testing service with their CI. Maintaining the security of SEVs is a significant concern to projects who rely on them.

Observation

CI tools are difficult and unwieldy to configure. This could be because of poor tooling or because developers do not have a good understanding of what they want their CI pipeline to be.

VIII. Implications

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

Developers CI started out as an "integration" technique, but its greatest value has been bringing automated testing into everyday practice. Developers should improve their CI process by improving the quality of their tests.

We found that most developers want their CI test suites to run in under 10 minutes. Developers should try their best to keep their test suites at that speed, so they can avoid a blocking step in their process as much as possible.

Multiple developers told us that they do not have confidence in projects that are not using CI. Developers who wish to attract the best talent should use CI. One interview participant even told us that since they were not able to expose their internal CI, they used a redundant external CI, just so that they would be taken seriously by the community.

Managers CI causes developers to give more value to automated tests. If managers wish to increase their test coverage, and get developers on their teams to care more about automated testing, they should adopt CI.

Since the underlying concept of CI is to have all the code in a single location, and since CI runs automated tests often, CI provides an ideal place to collect metrics about the quality of code, complexity of the codebase, etc. The data can be beneficial to managers when they make decisions such as where in the code to pay down technical debt or which features they should implement next.

Managers may be hesitant to allow the developers to start using CI. However, since developers report projects with CI have better tests, better code, and are more productive, managers should encourage the use of CI.

Tool Builders The most common problem developers report is configuring CI tools. Tool developers should provide more intuitive CI configurations to ease the process of learning a new CI. Additionally, users suggested that a UI to make changes to the configuration would be very helpful. Tool Developers should focus resources on making CI tools easier to configure.

Another needed improvement is interoperability with other tools. Specifically tool builders should improve how secrets are handled by CI tools, as well as making it easier to interface with other systems, such as static checkers or code coverage tools.

Developers of small proprietary projects specifically mentioned that they feel underserved by the current CI tools

available. Tool Builders should consider offering lightweight versions of their tools, which are simpler to run and install for small projects.

Researchers Developers say spending less time debugging is one of the least common benefits of CI. They also say that debugging assistance is one of the most desired features currently missing in CI tools. This is a clear opportunity for researchers to find ways to help developers debug broken CI builds. Perhaps one way researchers can help developers is by applying previous fault localization techniques to CI.

Almost 80% of developers report having put effort into making their automated tests run faster. Researchers should apply to CI previous techniques for speeding up testing [39], as well as develop new techniques to help CI run automated tests faster.

Another observation from our work is that when asked about problems faced in CI, there was a fairly uniform distribution of answers. From this, we infer that there is a wide range of problems that researchers can focus on.

The second most common reason for not using CI was lack of familiarity among developers on the project. Given the benefits of CI, it should be included in the core curriculum by university educators, to prepare the next generation of developers.

IX. Related Work

Continuous Integration Studies Vasilescu et al. [5] performed a preliminary study of quality outcomes for open-source projects using CI. They found that CI does indeed correlate with positive outcomes, i.e., open-source projects with CI are more productive, without sacrificing quality. This confirms our results that developers experience more productivity when using CI on their projects.

We [4] presented recently a qualitative study of the costs, benefits, and usage of CI but only for *open-source* projects and with open-source developers. In contrast, this paper is focused on proprietary projects and mostly proprietary developers.

Other researchers have studied ways to improve CI. Vos et al. [40] Run CI tests even after deployment, to verify the production code. Staples et al. [41] describe Continuous Validation as a potential next step after CI/CD. Muslu et al. [42] ran the tests continuously in the IDE, even more often than running them in CI would entail.

Other work related to CI and automated testing includes generating acceptance tests from unit test [43], black box test prioritization [44], ordering of failed unit tests [45], generating automated tests at runtime [46], and prioritizing acceptance tests [47].

Continuous Delivery Continuous Delivery (CD), the automated deployment of software, is enabled by the use of CI. Leppänen et al. [48] conducted semi-structured interviews with 15 developers to learn more about CD. Their paper does not have any quantitative analysis and does not claim to provide generalized findings. Others have studied CD and MySQL schemas [49], CD at Facebook [50], and the tension between release speed and software quality when doing CD [51].

Developer Studies In this paper, we perform a study of developers to learn about their motivations, experiences, barriers, and unmet needs. Many other researchers have also studied developers, e.g., to learn how DevOps handles security [52], developers' debugging needs [53], how developers examine code history [54], and what barriers newcomers face in open-source projects [55].

Automated Testing In Section V we discuss code coverage as a metric for test quality. Miranda and Bertolino [56] explore new ways to calculate test coverage, using other projects. This could be used by CI to provide better coverage metrics to CI projects. Others [57] have used test coverage to develop coverage based test-selection techniques. Santos and Hindle [58] used Travis CI build status as proxy for code quality.

X. Threats to Validity

Verifiability *Can others replicate our results?* The interview script, survey questions, and raw data can be found on our website **http://cope.eecs.oregonstate.edu/ICSE17**.

Construct *Are we asking the right questions?* To answer our research questions, we used semi-structured interviews [22], which explore themes while letting participants bring up new ideas throughout the process. By allowing participants to have the freedom to bring up topics, we avoid biasing the interviews with our preconceived ideas of CI.

Internal *Did we skew the accuracy of our results with how we collected and analyzed information?* Interviews and surveys can be affected by bias and inaccurate responses. These could be intentional or unintentional. To mitigate these concerns, we followed established guidelines in the literature [18], [33], [59] for designing and deploying our survey. We ran iterative pilots for both studies and the survey, we kept the survey as short as possible, and offered a raffle to the survey participants.

External *Do our results generalize?* By interviewing selected developers, it is not possible to understand the entire developer population. To mitigate this, we attempted to recruit as diverse a population as possible, including 14 different companies, and a wide variety of company size and domain. We then used a survey with 523 participants to validate our responses. Because we recruited participants for the survey by advertising online, our results may be affected by self-selection bias.

XI. Conclusions

Software teams use CI for many activities, including to catch errors, make integration easier, and deploy more often. Our paper presented an extensive study, based on 16 interviews and 523 survey responses, of how developers, primarily on *proprietary* projects, perceive CI. On the positive, developers experience being more productive when using CI. However, despite the many benefits of CI, developers still encounter a wide variety of problems when using CI. We hope that this paper motivates researchers to tackle the hard problems that developers face with CI. CI is here to stay as a best practice in development, and we need continuous improvement (a different kind of "CI") of CI to realize its full potential.

XII. ACKNOWLEDGMENTS

We thanks Joel Spolsky, Brian Marick, Martin Fowler, Uncle Bob Martin, and Seb Rose for helping disseminate the survey. We also thank Amin Alipour, Denis Bogdanas, Mihai Codoban, Matthew Hammer, Kory Kraft, Sean McGregor, Shane McKee, Semih Okur, Cyrus Omar, Anita Sarma, and Sruti Srinivasa Ragavan, 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.

REFERENCES

[1] Puppet and DevOps Research and Assessments (DORA), "2016 State of DevOps Report," 2016.
[2] J. Humble, "Evidence and case studies."
[3] V. One, "10th Annual State of Agile Development Survey," 2016.
[4] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects," in *ASE*, 2016.
[5] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub," in *FSE*, 2015.
[6] E. S. Raymond, *The Cathedral & the Bazaar*.
[7] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE Transactions on Software Engineering*, vol. 30, 2004.
[8] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development," in *ICSE*, 2000.
[9] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. M. German, "Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub," in *ICSE*, 2015.
[10] "Continuous Integration," http://www.extremeprogramming.org/rules/integrateoften.html.
[11] M. Fowler, "Continuous Integration," 2006.
[12] "CruiseControl Home," http://cruisecontrol.sourceforge.net/.
[13] "Jenkins," https://jenkins.io/.
[14] "Travis CI - Test and Deploy Your Code with Confidence," https://travis-ci.org/.
[15] "Team Foundation Server | Visual Studio," https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx.
[16] "Docker on Windows Server 2016 Technical Preview 5," https://blog.docker.com/2016/04/docker-windows-server-tp5/.
[17] "Vagrant by HashiCorp," https://www.vagrantup.com/.
[18] S. Phillips, T. Zimmermann, and C. Bird, "Understanding and Improving Software Build Teams," in *ICSE*, 2014.
[19] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining Mental Models," in *ICSE*, 2006.
[20] K. Muşlu, C. Bird, N. Nagappan, and J. Czerwonka, "Transition from Centralized to Decentralized Version Control Systems," in *ICSE*, 2014.
[21] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How Do Software Engineers Understand Code Changes?" in *FSE*, 2012.
[22] F. Shull, J. Singer, and D. I. K. Sjøberg, Eds., *Guide to Advanced Empirical Software Engineering*. Springer London, 2008.
[23] http://cope.eecs.oregonstate.edu/ProprietaryCIUse.
[24] wercker, "Wercker - From code to containers," http://wercker.com/.
[25] "CruiseControl.NET," http://www.cruisecontrolnet.org/.
[26] "Continuous Integration and Delivery," https://circleci.com/.
[27] "TeamCity — Your 24/7 Build Engineer," https://www.jetbrains.com/teamcity/.
[28] "Xcode Server and Continuous Integration Guide: About Continuous Integration in Xcode," https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/xcode_guide-continuous_integration/.
[29] "Buildbot," http://buildbot.net/.
[30] "Continuous Integration and Deployment service for Windows developers," https://www.appveyor.com/.
[31] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.
[32] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding in-depth semistructured interviews problems of unitization and intercoder reliability and agreement," *Sociological Methods & Research*, vol. 42, 2013.
[33] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *CHASE*, 2013.
[34] N. Kerzazi and B. Adams, "Botched Releases," in *SANER*, 2016.
[35] N. Kerzazi, F. Khomh, and B. Adams, "Why Do Automated Builds Break?" in *ICSME*, 2014.
[36] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *ICSE*, 2013.
[37] "Coveralls.io - Test Coverage History and Statistics," https://coveralls.io/.
[38] "Selenium Testing, Mobile Testing, JS Unit Testing," https://saucelabs.com/.
[39] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *STVR*, vol. 22, 2012.
[40] T. Vos, P. Tonella, W. Prasetya, P. M. Kruse, A. Bagnato, M. Harman, and O. Shehory, "FITTEST," in *CSMR-WCRE*, 2014.
[41] M. Staples, L. Zhu, and J. Grundy, "Continuous Validation for Data Analytics Systems," in *ICSE*, 2016.
[42] K. Muşlu, Y. Brun, and A. Meliou, "Preventing Data Errors with Continuous Testing," in *ISSTA*, 2015.
[43] M. Jorde, S. Elbaum, and M. B. Dwyer, "Increasing Test Granularity by Aggregating Unit Tests," in *ASE*, 2008.
[44] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing White-box and Black-box Test Prioritization," in *ICSE*, 2016.
[45] M. Galli, M. Lanza, O. Nierstrasz, and R. Wuyts, "Ordering broken unit tests for focused debugging," in *ICSM*, 2004.
[46] S. Artzi, J. Dolby, S. H. Jensen, A. Møller, and F. Tip, "A Framework for Automated Testing of Javascript Web Applications," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 571–580.
[47] H. Srikanth, M. Cashman, and M. B. Cohen, "Test case prioritization of build acceptance tests for an enterprise cloud application," *Journal of Systems and Software*, vol. 119, 2016.
[48] M. Leppänen, S. Mäkinen, M. Pagels, V. P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The Highways and Country Roads to Continuous Deployment," *IEEE Software*, vol. 32, 2015.
[49] M. de Jong and A. van Deursen, "Continuous Deployment and Schema Evolution in SQL Databases," in *RELENG*, 2015.
[50] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous Deployment at Facebook and OANDA," in *ICSE*, 2016.
[51] G. Schermann, J. Cito, P. Leitner, and H. C. Gall, "Towards quality gates in continuous delivery and deployment," in *ICPC*, 2016.
[52] A. A. Ur Rahman and L. Williams, "Security Practices in DevOps," in *HotSos*, 2016.
[53] L. Layman, M. Diep, M. Nagappan, J. Singer, R. Deline, and G. Venolia, "Debugging Revisited," in *ESEM*, 2013.
[54] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey, "Software history under the lens," in *ICSME*, 2015.
[55] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming Open Source Project Entry Barriers with a Portal for Newcomers," in *ICSE*, 2016.
[56] B. Miranda and A. Bertolino, "Social Coverage for Customized Test Adequacy and Selection Criteria," in *AST*, 2014.
[57] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization," *Software Testing, Verification and Reliability*, vol. 25, 2015.
[58] E. A. Santos and A. Hindle, "Judging a Commit by Its Cover," in *MSR*, New York, NY, USA, 2016.
[59] I. Seidman, *Interviewing as Qualitative Research*. Teachers College Press, 2006.