Received ACM SIGSOFT Distinguished Paper Award at ESEC/FSE'17

Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility

Michael Hilton[1,2], Nicholas Nelson[1], Timothy Tunnell[3], Darko Marinov[3], Danny Dig[1]
[1]Oregon State University, USA
[2]Carnegie Mellon University, USA
[3]University of Illinois at Urbana-Champaign, USA
mhilton@cmu.edu,{nelsonni,digd}@oregonstate.edu,{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, tool builders invest in the wrong direction, and researchers miss many opportunities for improving the practice of software engineering.

In this paper, we present a qualitative study of the barriers and needs developers face when using CI. We conduct 16 semi-structured interviews with developers from different industries and development scales. We triangulate our findings by running two surveys. The *Focused Survey* samples 51 developers at a single company. The *Broad Survey* samples a population of 523 developers from all over the world. We find that when using and implementing CI, developers face trade-offs between speed and certainty (*Assurance*), between better access and information security (*Security*), and between more configuration options and greater ease of use (*Flexibility*). We present implications of these trade-offs for developers, tool builders, and researchers.

CCS CONCEPTS

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

KEYWORDS

Continuous Integration, Automated Testing

1 INTRODUCTION

Continuous integration (CI) systems automate the compilation, building, and testing of software. CI usage is widespread throughout the software development industry. For example, the "State of Agile" industry survey [51], with 3,880 participants, found half of the respondents use CI. The "State of DevOps" report [33], a survey of over 4,600 technical professionals from around the world, finds CI to be an indicator of "high performing IT organizations". We previously reported [19] 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).

ESEC/FSE'17, Paderborn, Germany
2017. 978-1-4503-5105-8/17/09…$15.00
DOI: 10.1145/3106237.3106270

Despite the widespread adoption of CI, there are still many unanswered questions about CI. In one study, Vasilescu et al. [50] show that CI correlates with positive quality outcomes. In our previous work [19], we examine the usage of CI among open-source projects on GitHub, and show that projects that use CI release more frequently than projects that do not. However, these studies do not present what barriers and needs developers face when using CI, or what trade-offs developers must make when using CI.

To fill in the gaps in knowledge about developers' use of CI, we ask the following questions: What needs do developers have that are unmet by their current CI system(s)? What problems have developers experienced when configuring and using CI system(s)? How do developers feel about using CI? Without answers to these questions, *developers* can potentially find CI more obstructive than helpful, *tool builders* can implement unneeded features, and *researchers* may not be aware of areas of CI usage that require further examination and solutions that can further empower practitioners.

To answer these questions, we employ complementary established research methodologies. Our primary methodology is interviews with 16 software developers from 14 different companies of all sizes. To triangulate [15] our findings, we deploy two surveys. The *Focused Survey* samples 51 developers at Pivotal[1]. The *Broad Survey* samples 523 participants, of which 95% are from industry, and 70% have seven or more years of software development experience. The interviews provide the content for the surveys, and the *Focused Survey* provides depth, while the *Broad Survey* provides breadth. Analyzing all this data, we answer four research questions:

RQ1: *What barriers do developers face when using CI?* (see §4.1)
RQ2: *What unmet needs do developers have with CI tools?* (see §4.2)
RQ3: *Why do developers use CI?* (see §4.3)
RQ4: *What benefits do developers experience using CI?* (see §4.4)

Based on our findings, we identify three trade-offs developers face when using CI. Other researchers [32, 34, 53] have identified similar trade-offs in different domains. We name these trade-offs *Assurance*, *Security*, and *Flexibility*.

Assurance describes the trade-off between increasing the added value that extra testing provides, and the extra cost of performing that testing. Rothermel et al. [34] identify this trade-off as a motivation for test prioritization.

Security describes the trade-off between increased security measures, and the ability to access and modify the CI system as needed. Post and Kagan [32] found a third of knowledge workers report security restrictions hinder their ability to perform their jobs. We observe this issue also applies to CI users.

[1]pivotal.io

Flexibility describes the trade-off that occurs when developers want systems that are both powerful and highly configurable, yet at the same time, they want those systems to be simple and easy to use. Xu et al. [53] identify the costs of over-configurable systems and found that these systems severely hinder usability. We also observe the tension from this trade-off among developers using CI.

In the context of these three trade-offs, we present implications for three audiences: *developers*, *tool builders*, and *researchers*. For example, *developers* face difficult choices about how much testing is enough, and how to choose the right tests to run. *Tool builders* should create UIs for CI users to configure their CI systems, but these UIs should serialize configurations out to text files so that they can be kept in version control. *Researchers* have much to bring to the CI community, such as helping with fault localization and test parallelization when using CI, and examining the security challenges developers face when using CI.

This paper makes the following contributions:
(1) We conduct exploratory semi-structured interviews with 16 developers, then triangulate these findings with a *Focused Survey* of 51 developers at Pivotal and a *Broad Survey* of 523 developers from all over the world.
(2) We provide an empirically justified set of developers' motivations for using CI.
(3) We expose gaps between developers' needs and existing tooling for CI.
(4) We present actionable implications that developers, tool builders, and researchers can build on.

The interview script, code set, survey questions, and responses can be found at http://cope.eecs.oregonstate.edu/CI_Tradeoffs

2 BACKGROUND

The idea of Continuous Integration (CI) was first introduced [6] 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) [3].

The core premise of CI, as described by Fowler [14], is that the more often a project integrates, the better off it is. CI systems are responsible for retrieving code, collecting all dependencies, compiling the code, and running automated tests. The system should output "pass" or "fail" to indicate whether the CI process was successful.

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

Changesets are a group of changes that a developer makes to the code. They may 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 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. One common way to trigger CI is when a commit is pushed to a repository.

For the CI to test the code without concern for previous data or external systems, it is important that CI runs in a clean environment. The automated build script should be able to start with a clean environment and build the product from scratch before executing tests. Many developers use containers (e.g., Docker[2]) to implement clean environments for builds.

An important step in the CI pipeline is confirming 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 perform other analyses, such as linting or evaluating test coverage.

The last step is to deploy the artifact. We found some developers consider deployment to be a part of CI, and others consider *continuous deployment* (CD) to be a separate process.

3 METHODOLOGY

Inspired by established guidelines [24, 28, 31, 41, 48], the primary methodologies we employ in this work are interviews with software developers and two surveys of software developers to triangulate [15] our findings.

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 [41]. Surveys are a quantitative technique that summarizes information over a larger sample size and thus provides broader results. Together, they provide a much clearer picture than either can provide alone.

We first use interviews to elicit developers experiences and expectations when working with CI, and we build a taxonomy of barriers, unmet needs, motivations, and experiences. We build a survey populating the answers to each question with the results of the interviews. We deploy this survey at Pivotal, a software and services company, that also develops a CI system, Concourse[3]. To gain an even broader understanding, we also deploy another survey via social media. The interview script, code set, survey questions, and the responses can be found on our companion site.

3.1 Interviews

We used semi-structured interviews "which include a mixture of open-ended and specific questions, designed to elicit not only the information foreseen, but also unexpected types of information" [41]. We developed our interview script by performing iterative pilots.

We initially recruited participants from previous research, and then used snowball sampling to reach more developers. 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 over eight years of development experience on average. We assigned each participant a subject number (Table 1). They all used CI, and a variety of CI systems, including Concourse[3], Jenkins[4], TravisCI[5], CruiseControl.NET[6], CircleCI[7], TeamCity[8], XCode Bots[9], Buildbot[10], Wercker[11], appVeyor[12], and proprietary CI systems. Each interview lasted between 30 and 60 minutes, and the participants were offered a US$50 Amazon gift card for participating.

[2]docker.io [3]concourse.ci [4]jenkins.io [5]travis-ci.org [6]cruisecontrolnet.org
[7]circleci.com [8]jetbrains.com/teamcity [9]developer.apple.com/xcode [10]buildbot.net
[11]wercker.com [12]appveyor.com

Table 1: Interview Participants

Subject	Exp.	Domain	Org. Size
S1	8 yrs.	Content Platform Provider	Small
S2	20 yrs.	Content Platform Provider	Small
S3	4 yrs.	Developer Tools	Large
S4	10 yrs.	Framework Development	Large
S5	10 yrs.	Content Management	Large
S6	10 yrs.	Computer Security Startup	Small
S7	5 yrs.	Framework Development	Small
S8	5 yrs.	Media Platform	Medium
S9	6 yrs.	Language Development	Medium
S10	9 yrs.	CI Platform Development	Medium
S11	6 yrs.	Software Development Consulting	Medium
S12	10 yrs.	CI Platform Development	Small
S13	12 yrs.	Telecommunications	Large
S14	5 yrs.	Software Development Consulting	Medium
S15	2 yrs.	Infrastructure Management	Medium
S16	8 yrs.	Cloud Software Development	Medium

The interviews were based on the research questions presented in Section 1. The following are some examples of the questions that we asked in the interview:

- Tell me about the last time you used CI.
- What tasks prompt you to interact with your 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 system?

We coded the interviews using established guidelines from the literature [35] and followed the guidance from Campbell et al. [7] on specific issues related to coding semi-structured interview data, such as segmentation, codebook evolution, and coder agreement.

The first author segmented the transcript from each interview by units of meaning [7]. The first two authors then collaborated on coding the segmented interviews, using the negotiated agreement technique to achieve agreement [7]. Negotiated agreement is a technique where both researchers code a single transcript and discuss their disagreements in an effort to reconcile them before continuing on. We coded the first eight interviews together using this negotiated agreement technique. Because agreement is negotiated along the way, there is no inter-rater agreement number. After the eighth interview, the first and second author independently coded the remaining interviews. 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.

3.2 Survey

We created a survey with 21 questions to quantify the findings from our semi-structured interviews. The questions for the survey were created to answer our research questions, focusing on what *benefits*, *barriers*, and *unmet needs* developers have when using CI.

The survey consisted of multiple choice questions, with a final open-ended text field to allow participants to share any additional information about CI. The answers for these multiple choice questions were populated from the answers given by interview participants. We ensured completeness by including an "other" field where appropriate. To prevent biasing our participants, we randomized the order of answers in multiple-choice questions.

Focused Population We deployed our survey to a focused population of developers at Pivotal. Pivotal embraces agile development and also sponsors the development of Concourse CI. We sent our survey via email to 294 developers at Pivotal, and we collected 51 responses for a response rate of 17.3%. All respondents from Pivotal reported using CI.

Broad Population We believe there are many voices among software developers, and we wanted to hear from as many of them as possible. We chose our sampling method for the *Broad Survey* to reach as many developers as possible. We recruited participants by advertising our survey on social media (Facebook, Twitter, and reddit). As with all survey approaches, we were forced to make certain concessions [5]. When recruiting participants online, we can reach larger numbers of respondents, but in doing so, results suffer self-selection bias. To maximize participation, we followed guidelines from the literature [42], including keeping the survey as short as possible, and raffling one US$50 Amazon gift card to survey participants.

We collected 523 complete responses, and a total of 691 survey responses, from over 30 countries. Over 50% of our participants had over 10 years of software development experience, and over 80% had over 4 years experience.

4 ANALYSIS OF RESULTS

4.1 Barriers

We answer *What barriers do developers face when using CI?* (**RQ1**)

We collected a list of barriers which prevent of hinder adoption and use of CI that our interview participants reported experiencing when using CI. 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 choose the three most common.

Table 2: Barriers developers encounter when using CI

Barrier	Broad	Focused
B1 Troubleshooting a CI build failure	50%	64%
B2 Overly long build times	38%	50%
B3 Automating the build process	34%	26%
B4 Lack of support for the desired workflow	31%	42%
B5 Setting up a CI server or service	27%	29%
B6 Maintaining a CI server or service	27%	40%
B7 Lack of tool integration	26%	12%
B8 Security and access controls	21%	14%

B1 Troubleshooting a CI build failure. When a CI build fails, some participants begin the process of identifying why the build failed. Sometimes, this can be fairly straightforward. However, for some build failures on the CI server, where the developer does not have the same access as they have when debugging locally, troubleshooting the failure can be quite challenging. S4 described one such situation:

If I get lucky, I can spot the cause of the problem right from the results from the Jenkins reports, and if not, then it becomes more complicated.

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[13], a service for automated testing of web pages, in conjunction with their CI. When a test fails on Sauce Labs, there is a recording that the developers can watch to determine exactly how their test failed. Another participant described how Wercker 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.

B2 Overly long build times. Because CI must confirm that the current changeset is integrated correctly, it must build the code and run automated tests. This is a blocking step for developers, because they do not want to accept the changeset until they can be certain that it will not break the build. If this blocking step becomes too long, it reduces developers' productivity. Many interview participants reported that their build times slowly grow over time, e.g., according to S10:

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 worst 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, dependency issues during the build process, and adding different styles of tests (e.g., acceptance tests) to the CI builds.

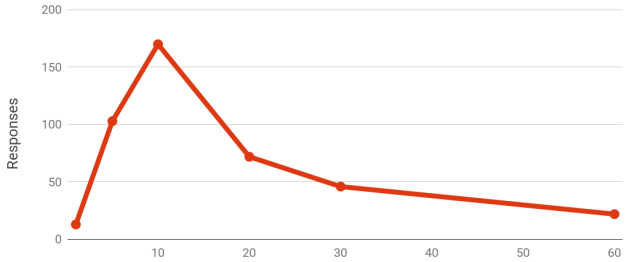

Figure 1: Maximum acceptable build time (minutes)

To dig a little deeper, we examined in-depth what developers meant by overly long build times. S9 said:

My favorite way of thinking about build time is basically, you have tea time, lunch time, or bedtime. Your builds should run in like, 5-ish 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 8-ish hours, however long it takes to go and come back the next day.

Fowler [14] suggests most projects should try to follow the XP guideline of a 10-minute build. When we asked our Broad Survey participants what is the maximum acceptable time for a CI build to take, the most common answer was also 10 minutes, as shown in Figure 1.

Many of our interview participants reported having spent time and effort reducing the build time for their CI process. S15 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 too long to get the feedback.

When we asked our survey participants, 96% of Focused Survey participants and 78% of Broad Survey participants said they had actively worked to reduce their build times. This shows long build times are a common barrier faced by developers using CI.

B3 Automating the build process. CI systems automate the manual process that developers previously followed when building and testing their code. The migration of these manual processes to automated builds requires that developers commit time and resources before the benefits of CI can be realized.

B4 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, S2 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.

B5 Maintaining a CI server or service. This barrier is similar to *N1 Easier configuration of CI servers or services*; see section 4.2.

B6 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 found that large commercial projects, while very complex, often have the resources to hire dedicated personnel to manage their CI pipeline. However, developers on small proprietary projects do not have the resources to afford CI as a service, nor do they have the hardware and expertise needed to setup CI locally. S9, who develops an app available on the Apple 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. ... this 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.

While some of these concerns can be addressed by tool builders creating tools targeted for smaller scale developers, more research is needed to determine how project size impacts the usage of CI.

B7 Lack of tool integration. This barrier is similar to *N2 Better tool integration*; see section 4.2.

B8 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 company 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

[13] saucelabs.com

developers working on company driven open-source projects, this can also be a concern. S9 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. S16, 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 in their project.

Observation

Developers encounter increased complexity, increased time costs, and new security concerns when working with CI. Many of these issues are side-effects of implementing new CI features such as more configurability, more rigorous testing, and greater access to the development pipeline.

4.2 Needs

We next answer *What unmet needs do developers have with CI tools?* (**RQ2**) In addition to describing problems they encounter when using CI, our interview participants also described gaps where CI was not meeting their needs.

Table 3: Developer needs unmet by CI

Need	Broad	Focused
N1 Easier configuration of CI servers or services	52%	32%
N2 Better tool integration	38%	17%
N3 Better container/virtualization support	37%	27%
N4 Debugging assistance	30%	30%
N5 User interfaces for modifying CI configurations	29%	20%
N6 Better notifications from CI servers or services	22%	25%
N7 Better security and access controls	16%	32%

N1 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. S8, 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...*

N2 Better tool integration. Our interview participants told us that they would like their CI system to better integrate with other tools. For example, S3 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.*

Additionally, 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".

N3 Better container/virtualization support. One core concept in 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, e.g., Docker. However, there are still times when the build fails, and in doing so, breaks the CI server. S15 explained:

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

S3 had experienced the same issues and had resorted to building Docker containers inside other Docker containers to ensure that everything was cleaned up properly.

N4 Debugging assistance. When asked about how they debug test failures detected by their CI, most of our participants told us that they get the output logs 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. S7 suggested that they would like their CI server to diff the output from the previous run and hide all the output which remained unchanged. S15, who worked for a large company, had developed an in-house tool to do exactly this, to help developers find errors faster by filtering the output to only show changes from the previous CI run.

N5 User interfaces for modifying CI configurations. Many participants described administering their CI tools via configuration scripts. However, participants expressed a desire to make these configuration files editable via a user interface, which they felt would be easier. S3 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.*

Additionally, multiple participants also added "Bad UI" as a free-form answer to the question about problems experienced with CI. Developers want to be able to edit their configuration files via user interfaces, but they also want to be able to commit these configurations to their repository. Our interview participants told us they want to commit the configurations, because then when they fork a repository, the CI configurations are included with the new fork as well.

N6 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, S7 said that he will routinely receive 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 S10 who reads through them every morning, to refresh his memory of where he left off the day before.

N7 Better security and access controls. This need is similar to *B8 Security and access controls*; see section 4.1.

Observation

Developers want CI to be both a highly-configurable platform, and simple to setup and maintain. This creates tension because adding configurability increases complexity, whereas simplification necessarily seeks to reduce complexity.

4.3 Motivations

We next answer *Why do developers use CI?* (**RQ3**). We identified developer motivations from the interviews.

Table 4: Developers' motivation for using CI

Motivation	Broad	Focused
M1 CI helps us catch bugs earlier	75%	86%
M2 CI makes us less worried about breaking our builds	72%	82%
M3 CI provides a common build environment	70%	78%
M4 CI helps us deploy more often	68%	75%
M5 CI allows faster iterations	57%	76%
M6 CI makes integration easier	57%	75%
M7 CI can enforce a specific workflow	40%	51%
M8 CI allows testing across multiple platforms	29%	73%

M1 CI helps catch bugs 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 [22] reported that 50% of all post-release failures were because of bugs. We would expect 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, S3 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.*

M2 Less worry about breaking the build. Kerzazi et al. [23] 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, S3 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.*

S2 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.*

M3 Providing a common build environment. One challenge developers face is ensuring that 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 in their team if the code does not build on the CI server, then the build is considered broken, regardless of how it behaves on an individual developer's machine. For example, S5 said:

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

M4 CI helps projects deploy more often. Our previous work [19] found that open-source projects that use CI deploy twice as often as projects that do not use CI. In our interviews, developers told us that they feel that CI helped them deploy more often. Additionally, developers told us that CI enabled them to have shorter development cycles than they otherwise would have, even if they did not deploy often for business reasons. For example, S14 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, it's 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.*

M5 CI allows faster iterations. Participants told us that running CI for every change allows them to quickly identify when the current changeset will break the build, or will cause problems in some other location(s) 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. S15 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.*

M6 CI makes integration easier. Initially, CI was presented as a way to avoid painful integrations [14]. However, while developers do think CI makes integration easier, it is not the primary reason that motivates developers to use CI. For many developers, they see their VCS as the solution to difficult integrations, not the CI.

M7 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 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 a code breaks the tests or the builds, without having to download the code and check 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 [2]. S16 told us that he was pretty sure that before they added CI to their project, contributors were not running the tests routinely.

M8 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, e.g., S16 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.*

Nevertheless, 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 which substantial improvements could be made to CI to provide additional support.

Observation

Developers use CI to guarantee quality, consistency, and viability across different environments. However, adding and maintaining automated tests causes these benefits to come at the expense of increased time and effort.

4.4 Experiences

We next answer the research question *What benefits do developers experience using CI?* (**RQ4**)

Devanbu et al. [11] found that developers have strongly held beliefs, often based on personal experience more than research results, and that practitioner beliefs should be given due attention. In this section we present developers' beliefs, gathered from interviews, about using CI. Our results show developers are very positive about the use of CI.

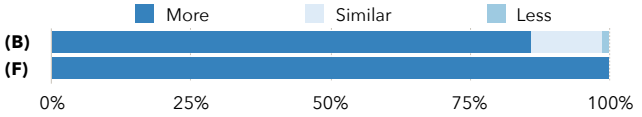

Figure 2: Do developers on projects with CI give (more/similar/less) value to automated tests?
Data Sources: (B)road Survey, (F)ocused Survey

E1 Developers believe projects with CI give more value to automated tests. Several participants told us that before using CI, although developers would write unit tests, they often would not be run, and developers did not feel that writing tests was worth the effort. S11 related:

> *Several 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 changeset, and the whole team becomes aware when the tests catch an error that otherwise would have made it into the product. S16 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.*

E2 Developers believe 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 write higher quality tests.

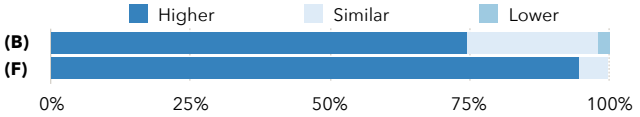

Figure 3: Do projects with CI have (higher/similar/lower) test quality?
Data Sources: (B)road Survey, (F)ocused Survey

Several participants claimed that using CI resulted in higher test coverage, which they equate with higher quality tests. For example, S8 stated:

> *... We jumped the coverage from a single digit to 50% of the code base in one year.*

To confirm this, we asked the same question of survey participants. Figure 3 shows that the survey participants overwhelmingly agree that projects with CI have higher quality tests.

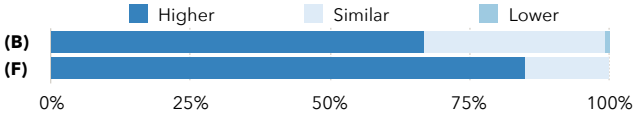

Figure 4: Do projects with CI have (higher/similar/lower) code quality?
Data Sources: (B)road Survey, (F)ocused Survey

E3 Developers believe projects that use CI have higher code quality. Developers believe that using CI leads to 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. S10 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.*

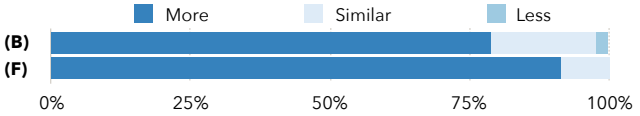

Figure 5: Are developers on projects with CI (more/similar/less) productive?
Data Sources: (B)road Survey, (F)ocused Survey

E4 Developers believe projects with CI are more productive. According to our interview participants, 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. S2 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. S16 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.

Observation

Some perceived benefits of CI are also benefits from automated testing, test-suite maintenance, and adherence to project standards. However, developers attribute these benefits to CI.

5 DISCUSSION

In this section, we discuss the trade-offs developers face when using CI, the implications of those trade-offs, and the differences between our two surveys.

5.1 CI Trade-Offs

As with any technology, developers who use CI should be aware of the trade-offs that arise when using that technology. We will look into three trade-offs that developers should be aware of when using CI: *Assurance*, *Security*, and *Flexibility*.

Assurance (Speed vs Certainty): Developers must consider the trade-off between speed and certainty. One of the benefits of CI is that it improves validation of the code (see M1, M2, and M9).

However, the certainty that code is correct comes at a price. Building and running all these additional tests causes the CI to slow down, which developers also considered a problem (see B2, M10). Ensuring that their code is correctly tested, but keeping build times manageable, is a trade-off developers must be aware of. Rothermel et al. [34] also identify this trade-off in terms of running tests as a motivation for test prioritization.

Security (Access vs Information Security): Information security should be considered by all developers. Developers are concerned about security when using CI (see B8, N7). This is important because a CI pipeline should protect the integrity of the code passing through the pipeline, protect any sensitive information needed during the build and test process (e.g., credentials to a database), as well as protect the machines that are running the CI system.

However, limiting access to the CI pipeline conflicts with developers' need for better access (see B1, N4). During our interviews, developers reported that troubleshooting CI build failures was often difficult because they did not have the same access to code running on a CI system, as they did when running it locally on their own machine. Providing more access may make debugging easier, but poses challenges when trying to ensure the integrity of the CI pipeline. Post and Kagan [32] examine this trade-off for knowledge workers, and found security restrictions hinder a third of workers from being able to perform their jobs.

Flexibility (Configuration vs Simplicity): Another trade-off that developers face is between the flexibility and power of highly configurable CI systems, and the ease of use that comes from simplicity. Developers wish to have more flexibility in configuring and using their CI systems (see B4, B7, N2, and N3). More flexibility increases the power of a CI system, while at the same time also increasing its complexity.

However, the rising complexity of CI systems is also a concern for developers (see B5, B6, N1, and N5). Developers' needs for more flexibility directly opposes the desire for more simplicity. Xu et al. [53] examine over-configurable systems and also found that these systems severely hinder usability.

5.2 Implications

Each of these three trade-offs leads to direct implications for *developers*, *tool builders*, and *researchers*.

Assurance (Speed vs Certainty)

Developers should be careful to only write tests that add value to the project. Tests that do not provide value still consume resources every CI build, and slow down the build process. As more tests are written over time, build times trend upward. Teams should schedule time for developers to maintain their test suites, where they can perform tasks such as removing unneeded tests [40], improving the test suite by filling in gaps in coverage, or increasing test quality.

Developers face difficult choices about the extent to which each project should be tested, and to what extent they are willing to slow down the build process to achieve that level of testing. Some projects can accept speed reductions because of large, rigorous tests. However, for other projects, it may be better to keep the test run times faster, by only executing some of the tests. While this can be done manually, developers should consider using advanced test selection/minimization approaches [4, 12, 16, 20, 54].

Tool builders can support developers by creating tools that allow developers to easily run subsets of their testing suites [54]. Helping developers perform better test selection can trade some certainty for speed gains.

Researchers should investigate the trade-offs between speed and certainty. Are there specific thresholds where the build duration matters more than others? Our results suggest that developers find it important to keep build times under 10 minutes. Researchers should find ways to give the best possible feedback to developers within 10 minutes. Another avenue for researchers is to build upon previous work [13] using test selection and test prioritization to make the CI process more cost effective.

Security (Access vs Information Security)

Developers should be cognizant of the security concerns that extra access to the CI pipeline introduces. This is especially a concern for developers inside companies where some or all of their code is open source. One interview participant told us that they navigate the dichotomy between security and openness by maintaining both an internal CI server that operates behind their company firewall, and using Travis CI externally. They cannot expose their internal CI due to confidentiality requirements, but they use external CI to be taken seriously and maintain a positive relationship with the developer community at large.

Tool Builders should provide developers with the ability to have more access to the build pipeline, without compromising the security of the system. One way of accomplishing this is could be to provide fine-grained account management with different levels of access, e.g., restricting less trusted accounts to view-only access of

build results, and allowing trusted accounts to have full access to build results and management features in the CI system.

Researchers should explore the security challenges that arise when using CI. Although CI aims at automating and simplifying the testing and validation process, the increased infrastructure provides additional attack vectors that can be exploited. The security implications of CI require more thorough examination by security researchers in particular. Researchers should also examine the feasibility of creating systems that allow developers to safely expose their CI systems without compromising their security.

Flexibility (Configuration vs Simplicity)

Developers should recognize that custom development processes bring complexity, increase maintenance costs, installation costs, etc. They should consider adopting convention over configuration if they want to reduce the complexity of their CI system. Developers should strive to keep their processes as simple as possible, to avoid adding unneeded complexity.

Developers should consider the long-term costs of highly complex custom CI systems. If the CI becomes overly complex, and the administration is not shared among a team, there is a vulnerability to the overall long-term viability if the maintainer leaves the project. Developers should also consider the long-term maintenance costs when considering adding complexity to their CI pipeline.

Tool Builders must contend with developers that want expanded UIs for managing the CI pipeline, as well as having the underlying configurations be captured by version-control systems. Tool Builders should create tools that allow for UI changes to configurations, but also output those configurations in simple text files that can be easily included in version control.

Researchers should collect empirical evidence that helps developers, who wish to reduce complexity by prioritizing convention over configuration, to establish those conventions based on evidence, not on arbitrary decisions. Researchers should develop a series of empirically justified "best practices" for CI processes. Also, developers who use CI believe strongly that CI improves test quality, and that CI makes them more productive. Researchers should evaluate whether these claims are indeed true.

5.3 Focused (Pivotal) vs Broad Survey Results

We deployed the Focused Survey at a single company (Pivotal), and the Broad Survey to a large population of developers using social media. After performing both surveys, we discussed the findings with a manager at Pivotal, and these discussions allowed us to develop a deeper understanding of the results.

Flaky Tests The survey deployed at Pivotal contained 4 additional questions requested by Pivotal. One question asked developers to report the number of CI builds failing each week due to true test failures. Another question asked developers to estimate the number of CI builds failing due to non-deterministic (flaky) tests [27]. Figure 6 shows the reported number of CI build failures because of flaky tests, as well as failures due to true test failures. There was no significant difference between the two distributions (Pearson's Chi-squared test, p-value = 0.48), suggesting that developers experienced similar numbers of flaky and true CI failures per week.

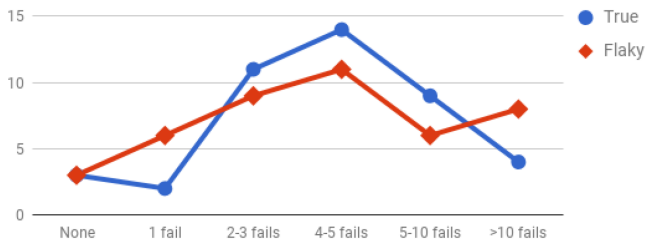

Figure 6: Flaky vs True test failures reported by Pivotal developers (N=42)

However, for the largest category, >10 fails a week, there were twice as many flaky failures as true failures.

When we discussed our findings with the manager at Pivotal, he indicated this was the most surprising finding. He related that at Pivotal, they have a culture of trying to remove flakiness from tests whenever possible. That claim was supported by our survey response, where 97.67% of Pivotal participants reported that when they encounter a flaky test, they fix it. Nevertheless, our participants reported that CI failures at Pivotal were just as likely to be caused by flaky tests as by true test failures.

Build Times Focused Survey respondents indicated that their CI build times typically take "greater than 60 minutes". This is in contrast with the "5-10 minutes" average response from respondents in the Broad Survey. This difference can also be observed in the acceptable build time question, in which Focused Survey respondents selected "varies by project" most often compared to the Broad Survey respondents that selected "10 minutes" as the most commonly acceptable build time.

Pivotal management promotes the use of CI, and its accompanying automation, for as many aspects of their software development as possible. According to the manager at Pivotal, the difference in responses for actual and acceptable build times can be explained by the belief that adhering to test-driven development results in significantly more unit tests, but for Pivotal, the extra testing is worth the longer CI build times. The manager also suggested that the addition of multiple target platforms in CI builds will also necessarily increase build times. Therefore, at Pivotal, while they seek to reduce those times whenever possible, they accept longer build times when necessary.

Maintenance Costs Focused Survey respondents reported experiencing "troubleshooting a CI build failure", "overly long CI build times", and "maintaining a CI server or service" more often than the Broad Survey respondents. When asked about this difference, the manager at Pivotal indicated that they actively promote a culture of process ownership within their development teams, so the developers are responsible for maintaining and configuring the CI services that they use. They also said that the CI systems they use are more powerful and complex than other CI systems, resulting in a more complicated setup, but provides more control over the build process.

6 THREATS TO VALIDITY

Replicability *Can others replicate our results?* Qualitative studies in general are very difficult to replicate. We address this threat by conducting interviews, a focused survey at a single company, and a large-scale survey of a broad range of developers. The interview script, code set, survey questions, and raw data can be found on our companion site. We cannot publish the transcripts because we told the interview participants we would not release the transcripts.

Construct *Are we asking the right questions?* To answer our research questions, we used semi-structured interviews [41], which explore themes while also 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. We gave interviewees gift cards for their participation and offered the survey participants the chance to win a gift card, which could bias our results.

To mitigate these concerns, we followed established guidelines in the literature [31, 39, 42] for designing and deploying our survey. We ran iterative pilots for both studies and the surveys, and we kept the surveys as short as possible.

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 domains. We then validate our responses using the Focused Survey with 51 responses, and the Broad Survey with 523 responses from over 30 countries. Because Pivotal is a company which builds a CI tool, the results could be biased in favor of CI. To mitigate this, we widely recruited participants for the Broad Survey. However, because we recruited participants for the Broad Survey by advertising online, our results may be affected by self-selection bias.

7 RELATED WORK

Continuous Integration Studies Vasilescu et al. [50] performed a preliminary quantitative study of quality outcomes for open-source projects using CI. Our previous work [19] presented a quantitative study of the costs, benefits, and usage of CI in open-source software. These studies do not examine barriers or needs when using CI, nor do they address the trade-offs developers must contend with. In contrast to these studies, we develop a deep understanding of the the barriers and unmet needs of developers through interviews and surveys. We also discover trade-offs users face when using CI.

Debbiche et al. [10] present a case study of challenges faced by a telecommunications company when adopting CI. They present barriers from a specific company, but provide no generalized findings and do not address needs, experiences, or benefits of CI.

Other researchers have studied ways to improve CI. Ståhl and Bosch [44] study automated software integration, a key building block for CI. Elbaum et al. [13] examined the use of regression test selection techniques to increase the cost-effectiveness in CI. Vos et al. [52] propose running CI tests even after deployment, to check the production code. Muşlu et al. [29] ran tests continuously in the IDE, even more often than in CI. Staples et al. [45] describe Continuous Validation as a potential next step after CI/CD.

Other work related to CI and automated testing includes generating acceptance tests from unit tests [21], black-box test prioritization [18], ordering of failed unit tests [17], generating automated tests at runtime [1], and prioritizing acceptance tests [43].

Continuous Delivery Continuous Delivery (CD), the automated deployment of software, is enabled by the use of CI. Olsson et al. [30] performed a case study of four companies transitioning to continuous delivery. They found some similar barriers when transitioned to CD as we find for CI, including *automating the build process* (B3), *lack of support for desired workflow* (B4), and *lack of tool integration* (B7).

Leppänen et al. [26] 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 [9], CD at Facebook [37], and the tension between release speed and software quality when doing CD [38].

Developer Studies We perform a study of developers to learn about their barriers, unmet needs, motivations, and experiences. Many other researchers have also studied developers, e.g., to learn how DevOps handles security [49], developers' debugging needs [25], and how developers examine code history [8].

Automated Testing Previous work has examined the intertwined nature of CI and automated testing. Stolberg [46] and Sumrell [47] both provide experience reports of the effects of automating tests during transitions to CI. Santos and Hindle [36] used Travis CI build status as proxy for code quality.

8 CONCLUSIONS AND FUTURE WORK

Software teams use CI for many activities, including to catch errors, make integration easier, and deploy more often. Developers also experience being more productive when using CI. Despite the many benefits of CI, developers still encounter a wide variety of problems with CI. We hope that this paper motivates researchers to tackle the hard problems that developers face with CI.

For example, future work should examine the relationship between developers' desired and actual build times when using CI. Another area that we identified for future work is a deeper analysis into flaky tests. Flaky test identification tools could automatically detect flaky tests to help developers know if CI failures are due to flaky tests or legitimate test failures. CI is here to stay as a development practice, and we need *continuous improvement ("CI" of a different kind) of CI* to realize its full potential.

ACKNOWLEDGMENTS

We thank Martin Fowler, Brian Marick, and Joel Spolsky for promoting the *Broad Survey*, and Matthew Kocher for all the help with the *Focused Survey* at Pivotal Labs. We also thank Amin Alipour, Andrew Begel, Souti Chattopadhyay, Mihai Codoban, Matt Hammer, Sean McGregor, Cyrus Omar, Anita Sarma, and the anonymous reviewers for their valuable comments on earlier versions of this paper. This research was partially supported by NSF grants CCF-1421503, CCF-1438982, CCF-1439957, and CCF-1553741.

REFERENCES

[1] Shay Artzi, Julian Dolby, Simon Holm Jensen, Anders Møller, and Frank Tip. 2011. A Framework for Automated Testing of JavaScript Web Applications. In *ICSE*.

[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *ICSE*.

[3] Kent Beck. 1999. Embracing Change with Extreme Programming. *IEEE Computer* (1999).

[4] John Bible, Gregg Rothermel, and David S. Rosenblum. 2001. A Comparative Study of Coarse- and Fine-grained Safe Regression Test-selection Techniques. *TOSEM* (2001).

[5] Michael H Birnbaum. 2004. Human Research and Data Collection via the Internet. *Annual Review of Psychology* (2004).

[6] Grady Booch. 1990. *Object-Oriented Design with Applications*. Benjamin-Cummings Publishing Co., Inc.

[7] John L. Campbell, Charles Quincy, Jordan Osserman, and Ove K. Pedersen. 2013. Coding In-depth Semistructured Interviews: Problems of Unitization and Intercoder Reliability and Agreement. *Sociological Methods & Research* (2013).

[8] Mihai Codoban, Sruti Srinivasa Ragavan, Danny Dig, and Brian Bailey. 2015. Software History Under the Lens: A Study on Why and How Developers Examine It. In *ICSME*.

[9] Michael de Jong and Arie van Deursen. 2015. Continuous Deployment and Schema Evolution in SQL Databases. In *RELENG*.

[10] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. 2014. Challenges When Adopting Continuous Integration: A Case Study. In *PROFES*.

[11] Prem Devanbu, Thomas Zimmermann, and Christian Bird. 2016. Belief & Evidence in Empirical Software Engineering. In *ICSE*.

[12] Nima Dini, Allison Sullivan, Milos Gligoric, and Gregg Rothermel. 2016. The Effect of Test Suite Type on Regression Test Selection. In *ISSRE*.

[13] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for Improving Regression Testing in Continuous Integration Development Environments. In *FSE*.

[14] Martin Fowler. 2006. Continuous Integration. http://martinfowler.com/articles/continuousIntegration.html. (2006).

[15] Lisa M Given. 2008. *The SAGE Encyclopedia of Qualitative Research Methods*. SAGE Publications.

[16] Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. 2015. Practical Regression Test Selection with Dynamic File Dependencies. In *ISSTA*.

[17] Markus Gälli, Michele Lanza, Oscar Nierstrasz, and Roel Wuyts. 2004. Ordering Broken Unit Tests for Focused Debugging. In *ICSM*.

[18] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing White-box and Black-box Test Prioritization. In *ICSE*.

[19] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. In *ASE*.

[20] Sheng Huang, Jun Zhu, and Yuan Ni. 2009. ORTS: A Tool for Optimized Regression Testing Selection. In *OOPSLA*.

[21] Matthew Jorde, Sebastian Elbaum, and Matthew B. Dwyer. 2008. Increasing Test Granularity by Aggregating Unit Tests. In *ASE*.

[22] Noureddine Kerzazi and Bram Adams. 2016. Botched Releases: Do We Need to Roll Back? Empirical Study on a Commercial Web App. In *SANER*.

[23] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. 2014. Why Do Automated Builds Break? An Empirical Study. In *ICSME*.

[24] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *ICSE*.

[25] Lucas Layman, Madeline Diep, Meiyappan Nagappan, Janice Singer, Robert Deline, and Gina Venolia. 2013. Debugging Revisited: Toward Understanding the Debugging Needs of Contemporary Software Developers. In *ESEM*.

[26] Marko Leppänen, Simo Mäkinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V. Mäntylä, and Tomi Männistö. 2015. The Highways and Country Roads to Continuous Deployment. *IEEE Software* (2015).

[27] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An Empirical Analysis of Flaky Tests. In *FSE*.

[28] Kıvanç Muşlu, Christian Bird, Nachiappan Nagappan, and Jacek Czerwonka. 2014. Transition from Centralized to Decentralized Version Control Systems: A Case Study on Reasons, Barriers, and Outcomes. In *ICSE*.

[29] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. 2015. Preventing Data Errors with Continuous Testing. In *ISSTA*.

[30] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. 2012. Climbing the "Stairway to Heaven" – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *Euromicro SEAA*.

[31] Shaun Phillips, Thomas Zimmermann, and Christian Bird. 2014. Understanding and Improving Software Build Teams. In *ICSE*.

[32] Gerald V. Post and Albert Kagan. 2007. Evaluating Information Security Tradeoffs: Restricting Access Can Interfere with User Tasks. *Computers & Security* (2007).

[33] Puppet and DevOps Research and Assessments (DORA). 2016. 2016 State of DevOps Report. (2016).

[34] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test Case Prioritization: An Empirical Study. In *ICSM*.

[35] Johnny Saldaña. 2015. *The Coding Manual for Qualitative Researchers* (3 ed.). SAGE Publications.

[36] Eddie Antonio Santos and Abram Hindle. 2016. Judging a Commit by Its Cover: Correlating Commit Message Entropy with Build Status on Travis-CI. In *MSR*.

[37] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. 2016. Continuous Deployment at Facebook and OANDA. In *ICSE*.

[38] Gerald Schermann, Jürgen Cito, Philipp Leitner, and Harald C. Gall. 2016. Towards Quality Gates in Continuous Delivery and Deployment. In *ICPC*.

[39] Irving Seidman. 2006. *Interviewing as Qualitative Research: A Guide for Researchers in Education and the Social Sciences*. Teachers College Press.

[40] August Shi, Alex Gyori, Milos Gligoric, Andrey Zaytsev, and Darko Marinov. 2014. Balancing Trade-offs in Test-suite Reduction. In *FSE*.

[41] Forrest Shull, Janice Singer, and Dag I. K. Sjøberg (Eds.). 2008. *Guide to Advanced Empirical Software Engineering*.

[42] Edward Smith, Robert Loftin, Emerson Murphy-Hill, Christian Bird, and Thomas Zimmermann. 2013. Improving Developer Participation Rates in Surveys. In *CHASE*.

[43] Hema Srikanth, Mikaela Cashman, and Myra B. Cohen. 2016. Test Case Prioritization of Build Acceptance Tests for an Enterprise Cloud Application: An Industrial Case Study. *JSS* (2016).

[44] Daniel Ståhl and Jan Bosch. 2014. Automated Software Integration Flows in Industry: A Multiple-Case Study. In *ICSE Companion*.

[45] Mark Staples, Liming Zhu, and John Grundy. 2016. Continuous Validation for Data Analytics Systems. In *ICSE*.

[46] Sean Stolberg. 2009. Enabling Agile Testing through Continuous Integration. In *AGILE*.

[47] Megan Sumrell. 2007. From Waterfall to Agile – How does a QA Team Transition?. In *AGILE*.

[48] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. How Do Software Engineers Understand Code Changes? – An Exploratory Study in Industry. In *FSE*.

[49] Akond Ashfaque Ur Rahman and Laurie Williams. 2016. Security Practices in DevOps. In *HotSos*.

[50] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub. In *ESEC/FSE*.

[51] VersionOne. 2016. 10th Annual State of Agile Report. (2016).

[52] Tanja Vos, Paolo Tonella, Wishnu Prasetya, Peter M. Kruse, Alessandra Bagnato, Mark Harman, and Onn Shehory. 2014. FITTEST: A New Continuous and Automated Testing Process for Future Internet Applications. In *CSMR-WCRE*.

[53] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, You Have Given Me Too Many Knobs!: Understanding and Dealing with Over-Designed Configuration in System Software. In *ESEC/FSE*.

[54] Shin Yoo and Mark Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *STVR* (2012).