

# A Conceptual Replication of Continuous Integration Pain Points in the Context of Travis CI

David Gray Widder  
dwidder@cmu.edu  
Carnegie Mellon

Michael Hilton  
mhilton@cmu.edu  
Carnegie Mellon

Christian Kästner  
Carnegie Mellon

Bogdan Vasilescu  
vasilescu@cmu.edu  
Carnegie Mellon

## ABSTRACT

Continuous integration (CI) is an established software quality assurance practice, and the focus of much prior research with a diverse range of methods and populations. In this paper, we first conduct a literature review of 37 papers on CI pain points. We then conduct a conceptual replication study on results from these papers using a triangulation design consisting of a survey with 132 responses, 12 interviews, and two logistic regressions predicting TRAVIS CI abandonment and switching on a dataset of 6,239 GITHUB projects. We report and discuss which past results we were able to replicate, those for which we found conflicting evidence, those for which we did not find evidence, and the implications of these findings.

## CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools*.

## KEYWORDS

Continuous integration, open source software, replication

### ACM Reference Format:

David Gray Widder, Michael Hilton, Christian Kästner, and Bogdan Vasilescu. 2019. A Conceptual Replication of Continuous Integration Pain Points in the Context of Travis CI. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19)*, August 26–30, 2019, Tallinn, Estonia. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3338906.3338922>

## 1 INTRODUCTION

Continuous integration (CI) has enjoyed tremendous popularity as a quality assurance mechanism during software development, by automating the execution of builds, tests, and other tasks. CI adoption was primarily driven by practitioners,<sup>1</sup> but research has shown that CI practices have a positive effect on software quality and productivity [28, 69, 76].

Despite the widespread adoption of CI, it has long been established by contingency theory [50, 66] that a single “universal best practice” is unlikely, whatever the actual practice. Moreover, for CI specifically, the literature abounds with studies (we counted 37 papers; Section 3) that each touch on some CI pain points. For example, research has shown that it can take significant effort to

<sup>1</sup>[www.martinfowler.com/articles/continuousIntegration.html](http://www.martinfowler.com/articles/continuousIntegration.html)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-5572-8/19/08...\$15.00  
<https://doi.org/10.1145/3338906.3338922>

set up and customize CI infrastructure [27, 34], and reports from CI systems require effort to process and can cause unwanted interruptions [39], especially without developer buy-in and in the presence of frequent false positives from flaky tests and platform instabilities [37]. Bad experiences or frustration with a specific CI tool can turn developers away from CI as a practice, even when more customized tool solutions exist [86].

Given the number of studies, conducted using a multitude of methods, on diverse populations, we argue that it is the right time for a *thorough review of the pain points and context mismatches that turn people away from CI*. This can help practitioners adopt CI with realistic expectations and in a way that fits their needs, and researchers and tool builders focus on the most severe CI barriers.

In this paper we *review the CI literature* from the perspective of pain points to adoption and usage, and perform a mixed-methods *conceptual replication* [32, 65] of previously observed findings, on a new population (GITHUB open-source developers using TRAVIS CI), and using a robust study design. As particular strengths of our study design, we note: ① the mixed qualitative (survey with 132 developers and interviews with 12; Sec. 4) and quantitative (large-scale multivariate statistical modeling of trace data from 6,239 projects; Sec. 5) analyses, which enable us to *triangulate* our results; and ② the focus on CI *leavers* (rather than current CI users), *i.e.*, those who either switched the TRAVIS CI tool or abandoned the CI practice altogether, which, similarly to customer exit surveys in market research [70], enable us to identify the most acute of TRAVIS CI pain points, since they caused users to leave.

Our main results (Sec. 6), confirming past literature, are that many developers find troubleshooting build failures difficult, desire consistency in CI tools across their projects, find it difficult to use CI with complex tool setups including Docker or to use CI with unsupported languages, find long build times annoying, and find CI less useful without enough tests.

In summary, we contribute: (1) a literature review of general CI pain points; (2) an analysis of 132 survey responses about reasons for abandoning or switching TRAVIS CI; (3) regression models on a dataset of 6,239 GITHUB TRAVIS CI projects, testing observations from literature; and (4) a discussion of results and implications.

## 2 STUDY DESIGN

What are the major pain points that turn people away from CI? To answer this research question, we conduct a **conceptual replication** [32, 65], *i.e.*, we attempt to corroborate observations from past research using a different experimental design, on a different population. The importance of replication studies in software engineering is increasingly recognized.<sup>2</sup> Our *conceptual* replication, as opposed to an *exact* replication, represents a more robust design:

<sup>2</sup>*E.g.*, see the ROSE (Recognizing and Rewarding Open Science in Software Engineering) panel at FSE 2018: <https://tinyurl.com/y4m2uzsp>

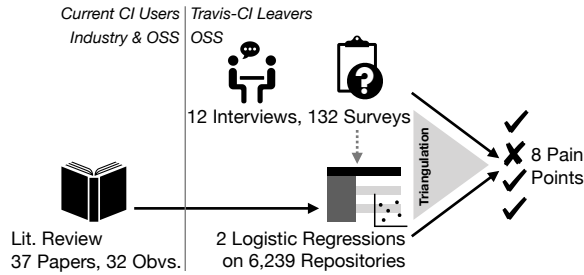


Figure 1: Overview of our mixed-methods study design.

we can be more confident in results that have been replicated on different populations and using different methods.

Whereas past work has studied CI in both open source and industry, we choose to focus on **open source projects** on GITHUB that used TRAVIS CI, for two reasons: First, open-source GITHUB projects are real-world projects representing a diversity of programming languages and application domains; and TRAVIS CI is the most popular CI service, used by as many as 90.1 % of CI-using GITHUB projects [28]. Second, both platforms make trace data publicly available (e.g., commits and build histories) [5, 20], which allows us to identify survey candidates and perform statistical analyses of repository data *at scale*.

Furthermore, where all prior work studied *current* CI users, e.g., surveying them about annoyances with their CI setup, we focus on the population of TRAVIS CI **leavers**, i.e., those for whom pain points were acute enough to *switch* tools or *abandon* CI altogether. Compared to CI adoption, leaving a CI service is arguably more deliberate: e.g., some of our survey respondents report that they considered alternatives and trade-offs when *switching* tools, but merely followed popular options during the initial adoption. This methodological change should allow us to better distinguish between minor CI annoyances and serious barriers, which may otherwise turn up during studies of current CI users. Leaving TRAVIS CI has the added benefit of being detectable from trace data.

Our study consists of several steps, outlined in Figure 1. First, we conduct a **literature review** of 37 papers discussing CI pain points, from which we distill 32 observations as candidates to replicate, of which 22 are relevant to our population (former TRAVIS CI users) (Sec. 3). Filtering the observations is necessary, as many prior industry-focused observations are out of scope in open-source (e.g., O2.d, *CI may create too many meetings*). However, for completeness we include all 32 observations in an online replication package.<sup>3</sup> Next, we proceed with a mixed-methods design that enables triangulation. On the one hand, we survey TRAVIS CI leavers (followed up with interviews to ensure deep understanding), inspired by **customer exit surveys** in market research [70, 79] (Sec. 4). On the other hand, we operationalize the selected observations and model them using **multivariate logistic regression** (Sec. 5). Multivariate regression modeling allows us to estimate the strength of the association between a specific factor (e.g., number of tests) and an outcome (e.g., abandoning TRAVIS CI) *while holding other variables fixed*, which makes our approach particularly robust to known confounding factors.

By combining the qualitative (survey) and quantitative (modeling) results, we can achieve **triangulation** [19] and reduce sources of methodological bias: On their own, surveys may be susceptible to human biases, e.g., social desirability effects [51], selection bias [67], hindsight bias [26]. Similarly, quantitative analyses of trace data may be impacted by skewed or missing data [49] and GITHUB-specific perils [33]; it is also harder to understand the motivations behind people’s decisions [6] from trace data alone. Instead, we evaluate agreement between both complementary methods [9], thereby reducing biases inherent in any single-method study.

### 3 LITERATURE REVIEW

There is a rich literature on CI adoption and use, with many papers mentioning pain points and mitigations. Here, we review 37 papers published since 2012 that discuss CI pain points, their effects, and ways to mitigate them (Table 1).

To find papers, we used the search term “Continuous Integration” in Google Scholar, in addition to our collective knowledge of CI literature and snowball sampling [38] from the resulting references. The identified papers used different research methods, including surveys of open-source CI users, case studies of industrial practices, and modeling the effects of CI adoption on outcomes like bugs reported and developer turnover. We read all papers and discarded those that do not identify any pain points (e.g., only study benefits). Seven papers that discuss or propose a technique to mitigate a pain point but did not provide direct evidence for the pain point’s existence were included as well (bottom partition of Table 1).

We conducted a *thematic synthesis* on the selected papers, in which one “categorizes studies and summarizes the findings by identifying major or recurrent themes within studies” [30], focusing on pain points. These major recurrent themes become the subsections of our literature review (📖 🛠️ ⚙️ 📊 📄). We enumerate all observations (e.g., “(O1.a)”), and list them again in the online replication package.<sup>3</sup>

**Information Overload** 📖. CI output logs can become verbose, making it hard to pinpoint bugs and their causes (O1.a) [27, 28, 39]. Also, information about a project may be spread out across many CI services or many sub-projects in an organization [52], making it difficult to answer overarching questions (O1.b). There is also a reported need for better notifications from CI (O1.c) [27], for customizing the outputs [8], and for pinpointing which build configuration caused a problem [69]. One industrial case study suggested risking performance creep, as smaller, frequent builds make performance changes harder to notice (O1.d) [63].

**Organizational Pain Points** 🏢. Organizational pain points have been reported, ranging from a general resistance to change (O2.a) [13, 40, 59], to the perception that CI practices may be incompatible with the team or company’s existing culture (O2.b) [12, 52, 63]. Industrial case studies show that management buy-in is crucial for CI success (O2.c) [13, 63]. Some suggest that CI can have negative effects on organizational culture, making them seem chaotic or disorganized [63], and force meetings to happen too frequently or prematurely (O2.d) [39]. Infrequent changes or a preexisting rigorous testing regime can create the perception that CI does not add additional value (O2.e,f) [28].

A lack of shared understanding of CI can cause conflicts between sites or between companies and their clients (O2.g) [52],

<sup>3</sup>zenodo.org/record/3265294

**Table 1: Literature review summary. Papers classified by studied population (🐙 OSS, 🏢 industry) and pain points identified (📢 information overload, 🏢 organizational, ⚙️ setup/config, ⌚ slow feedback, 🏢 testing deficiencies).**

Ref	Yr	Research Method(s)	Pop.	Pain Points
[77]	'18	Quant. analysis of quality metrics, 1,772 proj.	🐙	🏢
[12]	'18	Case study, 1 company	🏢	🏢
[57]	'18	Survey of 158 CI users	🐙	🏢 ⚙️ 🏢
[27]	'17	2 surveys of 51 and 523, 16 interviews	🐙	📢 ⚙️ 🏢
[60]	'17	Bivariate correlations, 14 Java projects	🐙	🏢 ⚙️ 🏢
[83]	'17	Bivariate correlations, 20 Java projects	🐙	🏢 ⚙️ 🏢
[4]	'17	Bivariate correlations, 2.6m builds	🏢	🏢 ⚙️ 🏢
[37]	'17	Comparison of averages, 61 projects	🐙	🏢 ⚙️ 🏢
[87]	'17	Time series of 77 projects, survey of 55	🐙	🏢 ⚙️ 🏢
[31]	'17	Bivariate cor., 3.6m builds, 1090 proj.	🐙	📢
[24]	'17	Bivariate correlations, 1283 projects	🐙	🏢
[43]	'17	Descriptive statistics, 1279 projects	🐙	🏢
[58]	'17	Survey of 158	🐙	🏢 ⚙️ 🏢
[53]	'17	Bivariate cor., topic modeling, 1283 proj.	🐙	📢
[28]	'16	Bivariate cor., 1.5m builds, surveys of 442	🐙	📢 🏢 ⚙️ 🏢
[69]	'16	Case studies, 3 companies	🏢	🏢
[63]	'16	Case studies, 2 companies	🏢	🏢 ⚙️
[21]	'16	Survey of 645, bivariate cor. on proj.	🐙	🏢
[8]	'15	Contrib. clustering on 20 proj., manual	🏢	📢
[82]	'15	Regression modeling on 103k PRs	🐙	🏢 ⚙️ 🏢
[35]	'15	Bivar. cor. of failures, 2 case studies	🏢	🏢 ⚙️
[39]	'15	27 interviews	🏢	📢 🏢 ⚙️ 🏢
[40]	'15	15 interviews	🏢	🏢 ⚙️ 🏢
[22]	'15	Survey of 749, analysis of projects	🐙	🏢
[76]	'15	Quant. modeling 246 proj. + PRs, issues	🐙	🏢
[75]	'14	Bivariate cor. on 223 repos	🐙	🏢
[13]	'14	Case study on 1 company, 13 interviews	🏢	🏢 ⚙️ 🏢
[56]	'13	33 interviews, 569 surveys	🏢	🏢 ⚙️ 🏢
[52]	'12	Case st. of 4 companies, 18 interviews	🏢	📢 🏢 ⚙️ 🏢
[41]	'18	Test suite prioritization technique	🐙	🏢
[18]	'18	Analysis of 9312 proj. + tool evaluation	🐙	🏢 ⚙️ 🏢
[2]	'17	Essay, constructed case study	NA	🏢 ⚙️ 🏢
[64]	'16	Appl. of model to past case studies	-	🏢 ⚙️ 🏢
[59]	'16	Comparison of features	-	📢 🏢 ⚙️ 🏢
[7]	'14	Built tool, user study with 9 tasks 16	Stud.	📢
[10]	'14	Built tool, evaluation on 23 projects	🏢	🏢 ⚙️ 🏢
[16]	'14	Tool building + eval. cost effectiveness	-	🏢 ⚙️ 🏢

and a lack of clarity about the goals of CI can lead to friction between teams [13]. Reports indicate interpersonal strain among teams when CI knowledge is distributed unevenly [39] or CI is adopted inconsistently [63], possibly because collaboration and information exchange is poorly supported (O2.h) [52, 58]. CI can be seen as taking a long time to perfect in an evolving organization [63], or as an unclear, unreachable, and unrealistic standard (O2.i) [39], perhaps leading to only intermittent usage [77].

Developers also experience negative effects of CI, including the perceived risk of reputation damage from breaking the build [40], causing reluctance to expose early versions of their work (O2.j) [13]. They disagree about whether it is acceptable to break the build, and what should constitute a broken build [58].

Some believe that CI can make it harder to onboard and retain developers (O2.k) [24, 58], though others argue that automated negative feedback is preferable to human negative feedback [71]. One industry study suggests that it can be hard to hire developers with CI expertise (O2.l) [63]; in open source, there is evidence that not all project members need to use or understand CI [43].

Broader organizational pain points can also exist: regulatory requirements or a conservative business model can prohibit frequent releases in some domains (e.g., medical devices) and thus can be seen to prohibit CI usage [40], because they demand that requirements are defined upfront (O2.m) [52]. Some company supported projects are reluctant to use a public CI service, e.g., for privacy or security reasons (O2.n) [27].

**Setup and Configuration Pain Points** ⚙️. CI can be hard to configure (O3.a) [18, 21, 27, 58, 60], especially when switching from an old build environment [27], testing components developed by a third party [39], or configuring a self-hosted CI server [2, 27]. The prevalence of anti-patterns in CI configuration files also suggests that users find configuring CI challenging [18]. Some would prefer graphical user interfaces for their CI tools (O3.b) [27].

Connecting other tools to CI can be difficult (O3.c) [13, 27], especially legacy and immature tools [13, 52]. Still, some GitHub projects attempt to use TRAVIS CI even if it does not support the programming language they use (O3.d) [4]. Surveys show that CI users dislike when tools force a specific workflow (O3.e).

**Slow Feedback** ⌚. Some projects find it hard to use CI because their build takes too long (O4.a) [27, 37, 39, 60], in many cases because of large test suites [16, 35, 40]. This makes it hard to get quick and continuous feedback, a hallmark benefit of CI.

Two studies suggest that CI might increase pull-request latency when maintainers wait for long CI builds before manual review (O4.b) [82, 87]. Many developers complain that long build times make it hard to debug CI builds (O4.c) [2, 4, 13, 52, 60], where some suggest test suite prioritization might shorten the feedback loop [16, 35, 41]. The cost of computational resources for CI has been reported as a pain point (O4.d) [58, 59], and some domains, such as DNA analysis, have datasets so large that CI remains impractical [2].

**Testing Deficiencies** 🏢. Insufficient tests are seen as a significant pain point for adopting CI (O5.a) [28, 39, 40, 52, 64]. Conversely, many see the value of CI, but see writing tests as challenging [27, 39, 40, 52, 64]. The difficulty of automating certain kinds of tests (e.g., GUI) or formerly manual processes is frequently reported as a significant pain point (O5.b) [13, 21, 39, 40, 56]. Qualitative research indicates that projects may lack a strong “testing culture” [56, 58], or that developers perceive writing tests as difficult [10, 56], not fun [56], or too time consuming (O5.c) [10, 39, 40]. Many common testing challenges can surface as pain points when adopting CI: estimating test coverage [69], agreeing on an appropriately strict level of coverage [64], or improving coverage metrics [64, 72].

*Flaky* tests are a pain point too (O5.d) [13, 37], as they make developers take CI build results less seriously [39, 60] or develop a false sense of security [58]. Difficulties recreating the production environment also diminish trust in CI build results (O5.e) [52].

## 4 SURVEY AND INTERVIEWS

Our survey design is based on the idea of *exit surveys* (cf. Sec. 2), conducted with developers who disabled TRAVIS CI. In this section, we detail our survey methodology. We present and discuss the survey results together with modeling results in Section 6.

**Survey Protocol.** We randomly selected 400 (expecting a 25% response rate) projects from our dataset of GitHub projects that deactivated TRAVIS CI (Section 5), cloned each repository locally, and parsed its commit log. We collected the names and emails

of the contributors who had most recently edited their project’s `.travis.yml` file, on the assumption that they could best answer questions about that project’s CI practices. We filtered out people whose emails were not also publicly available on their GITHUB profiles, and those contacted in our previous (to reduce survey fatigue). We then emailed the remaining contributors individually about their respective projects, asking the single question: “*Why did this project stop using TRAVIS CI, and what has the CI situation been like since then?*” We specifically designed the survey as a single question, so that participants could answer directly by replying to the email, to lower survey fatigue and the effort required to respond, and left it open ended to capture possible answers not found in the literature and avoid priming participants. We received 132 responses, a 33 % response rate.

**Interview Protocol.** While most survey responses were detailed enough to understand why a project had abandoned TRAVIS CI (mean 82 words per response), we also conducted 12 semi-structured interviews (recorded with participant consent, then transcribed), purposely sampled to stratify across different stated reasons. In these interviews, we discussed how people’s decision to abandon TRAVIS CI was made, whether there were any secondary reasons, and integrated aspects of participatory study design by discussing our operationalizations and integrating their suggestions [15, 61]. In the remainder of the paper, we refer to interview responses as extensions of the corresponding participant’s survey answer.

**Card Sorting.** We then open card sorted the survey responses, a standard approach to categorize and relate qualitative data [3, 27, 48]: Two authors card sorted the survey answers into cohesive categories in a collaborative process. Disagreements were discussed and, if needed, the categories were adjusted. The categorizations were then independently reviewed by the other authors. Survey responses yielded 9 themes, each described in detail in Section 6 and summarized in Table 4. Overall, 78 participants reported abandoning CI altogether vs. 54 switching tools. Note that some respondents stated multiple reasons for their decision to leave, so respondents may repeat across categories.

**Threats to Validity.** Without triangulation (Sec. 2), survey results may suffer from human biases such as social desirability effects [51], hindsight bias [26], or selection effects [67]. Note, some degree of subjectivity is inherent in qualitative research, and thus other researchers may interpret the same data differently [14]. To allow further replication, our full interview protocol is included in the online replication package.<sup>3</sup>

## 5 QUANTITATIVE ANALYSIS

For our quantitative analysis, we use logistic regression (cf. Sec. 2). **TRAVIS CI Leavers: Abandonment vs. Switching.** First, we assemble a dataset of GITHUB projects that stopped using TRAVIS CI between Dec 2011 and Aug 2017 by mining the TRAVIS CI API for projects that had originally signed up for the service, but have since disabled it; we refer to these projects as TRAVIS CI *leavers*. Of these, the overwhelming majority had ceased development activity shortly after disabling Travis, *i.e.*, had no commits beyond 30 days after Travis abandonment; we subsequently filtered these out as the *leaving* event cannot be disentangled from the termination of project development altogether.

As a novel operationalization, we detect *abandoners* and *switchers* as two different kinds of *leavers* based on the *commit status*

**Table 2: Descriptive statistics for our final dataset (1,087 abandoners, 352 switchers, 4,800 controls).**

Numeric variables	Mean	St. Dev.	Min	Max
Project age (days)	1680.74	392.43	1095	2999
Num. commits	340.02	1152.92	1	29303
Num. contribs	8.47	14.49	1	141
Num. jobs	2.69	3.02	1	57
Num. PRs	25.2	74	0	796
Last build duration	240.94	369.79	2	2976
Num. tests	43.46	326.64	0	16695
% Rebuilds	0.01	0.03	0	0.5
Binary variables	% True		% True	
Has long builds	22	New user	45	
Commercial users	22	Needs Docker	2	
Exposure to leavers	31	Low activity	93	
Lang. supported	91			

*context* of recent commits, which is a form of metadata attached to commits on GITHUB, used to visually indicate whether a service has performed a check on the commit and with which resulting status.<sup>4</sup> If leavers had any commit status context after their last recorded TRAVIS CI build, we label them as *switchers*; otherwise as *abandoners*. We identified 1,145 abandoners and 369 switchers.

**Control Group.** For modeling, we match our dataset of leavers with a control group of 4,902 *retainers* – projects that adopted TRAVIS CI around the same time and still use it. Starting from a larger dataset of projects using TRAVIS CI released by Zhao et al [87], we sample our control group using nearest-neighbor propensity score matching [11] on the TRAVIS CI adoption date; this ensures that the control group projects had the same CI options available when they first adopted TRAVIS CI. Note that leaving TRAVIS CI is relative rare compared to using it, hence our decision to down-sample current TRAVIS CI users when compiling the control group; in addition, a larger (but not too large) control group compared to leavers helps reduce sampling bias while maintaining relatively balanced groups. Table 2 shows descriptive statistics of the full dataset. An exploratory preliminary analysis of a similar dataset (without distinguishing between abandonment and switching) was performed in a recent short paper [78].

**Logistic Regression Modeling.** We built two logistic regression models with *abandonment* and *switching* as response, respectively. Binomial logistic regression models estimate the likelihood of a binary outcome given a set of predictors. The technique allows us to *explain* the likelihood of the dependent binary event (*abandoning* or *switching*) as a function of different factors, and to estimate the effect of each factor while holding the other variables fixed. For each factor (predictor), the model provides information on the statistical significance and strength of the association with the outcome (we report odds ratios, above or below 1); we also compute  $\eta^2$ , the share of the total variance explained by that predictor using an ANOVA analysis, with typical thresholds (*i.e.*, small:  $0.01 \leq \eta^2 < 0.06$ , medium:  $0.06 \leq \eta^2 < 0.14$ , or large:  $\eta^2 \geq 0.14$ ) [47].

As a precaution, for highly-skewed measures we filtered out the top 1 % of values as potential high-leverage points, to increase model robustness [55, 74], and we log-transformed variables as

<sup>4</sup>[developer.github.com/v3/repos/statuses/](https://developer.github.com/v3/repos/statuses/)

needed, to stabilize variance and reduce heteroscedasticity [29]. We further performed multicollinearity analysis, checking if the *Variance Inflation Factor* remained below 3 [1]. Past work has shown that general attributes of a project, such as the number of commits and contributors are associated with TRAVIS CI abandonment [78]; we control for these covariates explicitly.

**Operationalization.** While the survey (Sec. 4) can cover all kinds of pain points that our respondents report, the quantitative analysis is limited to pain points we can operationalize with GITHUB and TRAVIS CI trace data. As discussed in Sec. 2, we exclude seven observations as not relevant for our target population of leavers of TRAVIS CI in open source projects on GITHUB (O2.a, O2.d, O2.g, O2.h, O2.l, O2.m, O5.e); for example, we consider observation (O2.d) (*CI can be perceived as creating too many meetings*) to be less relevant in an open source context. In addition, for 10 of our observations, we did not find a plausible way to operationalize them at scale in trace data (O1.b, O1.c, O1.d, O2.b, O2.c, O2.i, O2.j, O2.k, O5.b, O5.c); for example, regarding observation O2.j (*some developers fear damage to their reputation when breaking the CI build, making them less willing to expose early versions of their work*), we may detect overly large commits but cannot detect the reason from public artifacts; similarly, operationalizing some other observations would require sophisticated code analysis techniques that cannot be easily generalized to multiple programming languages at scale (e.g., detecting GUI code (O5.b)). Finally, in some cases there are multiple observations that have the same observable symptoms, so we model them with the same factor; for example, (O4.a) users dislike long builds because they prohibit quick feedback, and (O4.b) long builds lead to high pull request latency, are both operationalized with the same factors: *last build duration* and *has long builds*.

This leaves us with 14 observations to operationalize with the following measures:

- *% Rebuilds*: We measure a *rebuild rate*, the percentage of a project's builds triggered by the same commit. These instances indicate that someone has manually re-triggered the build, which is suggestive of flaky builds or flaky infrastructure (O5.d) and may indicate difficulty troubleshooting a flaky build (O1.a).
- *Low activity*: We label a project as having *low activity* if there are any months without commits in the six months preceding the last build on TRAVIS CI, thus capturing projects that possibly get little benefit from CI due to low activity (O2.e).
- *Commercial users*: We label a project as having *commercial users* with a measure originally developed by Valiev *et al.* [73]: we parse the email address from the projects' commit logs and classify contributors based on the domain name of their email address as either "academic" (e.g., name@nyu.edu), "commercial" (e.g., name@facebook.com), "public" (e.g., name@gmail.com), or "unknown". Literature suggests that company supported projects are reluctant to use public CI infrastructure (O2.n).
- *Needs Docker*: We record whether repositories have a container build script ("Dockerfile") of the most popular container format *Docker* on the day before the project's last build (for those who left, this is also the date that they stopped using TRAVIS CI) arguing that such presence is a strong signal that the project intends to use containers. We experimented with detecting other development tools, in line with (O3.c), but Docker was the only one we were able to consistently detect at scale. More generally,

if attempts to configure container use lead to abandonment or switching, this may indicate configuration challenges (O3.a).

- *Language support*: We label a project as having *language support* by comparing the primary language of each project to the list of officially supported languages on the TRAVIS CI platform.<sup>5</sup> We use GITHUB's own detection of languages and consider the most used language as primary.<sup>6</sup> Based on the TRAVIS CI documentation, we classify C, C++, Go, Java, JavaScript, PHP, Python, and Ruby as supported. In our sample, primary languages without support or only with community support were C#, CSS, HTML, Objective-C, and Puppet. Literature suggests that lack of language support presents a significant barrier (O3.d).
  - *Last build duration*: We include the *last build duration*, as reported by the TRAVIS CI API as a predictor, as is the most straightforward way to operationalize annoyance resulting from long build times (O4.a), long builds leading to high PR latency (O4.b), long build times leading to more error prone configurations (O4.c), and the possibility that CI is impractical in data intensive domains (O4.d).
  - *Has long builds*: In contrast to above, we suspect nonlinearity in that moderate build times are acceptable but very long build times are frustrating and indicate a barrier (O4.a). To attempt to capture this effect, we classify projects as *having long builds* if they had builds in their history that are among the top 20 percent of build times, globally, in our dataset.
  - *Number of test files*: We consider test files to be those with the name "test[s]" or "spec[s]" in the directory or file name, in line with previous work [72]. We count the number of files rather than their size to better control for language differences. Literature suggests that projects with a preexisting testing culture are less likely to see benefit from CI (O2.f), as well as those with few or no tests (O5.a).
  - *New user*: We expect that technology barriers (O3.d, O3.c) especially affect new CI users, who then must choose whether to "hack" and attempt to manually add support, or switch early on to a platform with built-in support. To allow us to model this, we set a *new user* variable true if the project has fewer than 30 days between their first and last build or fewer than 30 builds total.
- Finally, we found a new pain point during our survey that was not covered by previous observations in literature (network effects), and we operationalize that pain point as well.
- *Exposure to leavers*: We check whether a project has contributors who have also contributed to projects on GITHUB that previously left TRAVIS CI. We only model exposure which occurs before a project's last TRAVIS CI build, *i.e.*, before the decision to leave TRAVIS CI for those in the *switcher* and *abandoner* treatment groups, because exposure after this point clearly cannot influence choices made in the past. This way, we test whether a project was potentially exposed to leaving decisions in other projects, by knowledge transferred through contributors.

**Threats to Validity.** When splitting abandoners from switchers, we cannot automatically detect switchers in our dataset if developers move to a private CI service that does not update the commit status context on GITHUB. However, our survey responses offer ground truth for a subset of our full dataset: respondents report

<sup>5</sup>docs.travis-ci.com/user/languages/

<sup>6</sup>help.github.com/articles/about-repository-languages/

**Table 3: Regression models for CI abandoners and switchers.**

	Switchers ( $R^2 = 28\%$ )		Abandoners ( $R^2 = 22\%$ )	
	OR Coeffs (Err.)	LR Chisq	OR Coeffs (Err.)	LR Chisq
(Intercept)	1161306.24 (2.45)***		4.57 (1.32)	
Project age	0.11 (0.34)***	46.10***	0.74 (0.18)	2.65
Num. commits	2.61 (0.09)***	139.97***	1.93 (0.04)***	225.67***
Num. contribs	1.24 (0.11)*	4.10*	1.51 (0.06)***	49.48***
Num. jobs	0.88 (0.13)	0.90	0.84 (0.07)*	6.07*
Num. PRs	0.73 (0.06)***	27.08***	0.57 (0.04)***	248.35***
Last build duration	0.57 (0.07)***	79.07***	0.67 (0.04)***	123.81***
Num. tests	0.95 (0.05)	0.79	0.84 (0.03)***	36.02***
Has long builds	1.17 (0.20)	0.61	1.26 (0.11)*	4.06*
Lang. supported	0.47 (0.21)***	12.11***	0.49 (0.12)***	33.10***
Commercial users	0.74 (0.18)	2.79	0.75 (0.11)**	7.30**
Exposure to leavers	1.83 (0.16)***	13.63***	1.49 (0.10)***	17.37***
New user	1.94 (0.19)***	15.25***	2.01 (0.10)***	51.46***
Needs Docker	1.03 (0.38)	2.70	0.80 (0.30)	1.31
% Rebuilds	1.19 (0.06)**	7.16**	1.15 (0.03)***	15.86***
Low activity	0.49 (0.24)**	8.59**	0.27 (0.15)***	71.79***
New usr: Docker	9.42 (0.81)**	8.20**	5.25 (0.54)**	9.93**

\*\*\* $p < 0.001$ , \*\* $p < 0.01$ , \* $p < 0.05$

whether they switch or abandon. When comparing responses with our automated detection, we estimate that our detection underreports switchers by 18%. We expect that the impact of this underreporting on the general observations in our models is small.

As usual, many operationalizations can only approximate the effects of the underlying pain points with data available; for example, we operationalize resource limitations of data-intensive projects (O4.d) by measuring the average build time, which captures only part of the pain point but is measurable automatically at scale. Despite careful inspection on a sample of our data to avoid systematic biases, we also cannot exclude all construct validity issues with our operationalized factors. We expect that by averaging over thousands of sampled projects through our regression models, our measures will reflect the intensity and directionality of underlying trends in the dimensions we study. When we triangulate in conjunction with our survey, which does not suffer from this bias (but from others, e.g., hindsight bias, which does not threaten the model) we build confidence in our final, triangulated results.

Our dataset contains a large sample of GITHUB projects for which we could detect recent disabling of TRAVIS CI. While TRAVIS CI is currently the most popular cloud-based CI service, care must be taken when generalizing results to other CI services, or beyond open source projects on GITHUB.

## 6 RESULTS

We list the eight themes that emerged from our survey with corresponding survey participants in Table 4 and report our regression models (separately for switchers and abandoners) in Table 3.

Triangulating our results and comparing them to observations from previous literature, we discuss key reasons for abandoning or switching CI service in our target population. Specifically, we found eight factors that have strong support from either method in our replication, summarized in Table 4.

When appropriate, we report effect sizes in two ways: the approximate effect of a 1% increase in numerical predictors or of

true cases for binary predictors [80], and also amount of deviance explained by that predictor (Table 3), relative to other variables.

### 6.1 Long Build Times ☹

Fifteen respondents indicated leaving TRAVIS CI because of build speed issues (A100, A104, S35, S4, S59, S75, A80, S99, S125, S128, S26, S27, S5, S51, S62), and five others reported similar resource constraints including memory, disk space, and available build time (A14, S26, A79, S81, A23), together representing the second most common reason cited in our survey, and concurring with observations O4.a [16, 27, 35, 37, 39, 40, 60], O4.b [82, 87], O4.c [2, 4, 13, 52, 60], and O4.d [2, 58, 59]. Build speed issues manifested in different ways. Some complained about long build times and thus slow feedback with similar builds taking very different lengths of time (S51), others were frustrated by long queue times (S5, S125); e.g., S125 complained “We stopped using TRAVIS CI because it was working too slow for us. After pushing new changes, the build would wait up to 30 minutes in the queue.” Several respondents explicitly indicated that they switched to a different CI service because the target CI service offered faster builds (S27, S35, S51, S62, S75, S128) or the opportunity to parallelize elements of the build process, which TRAVIS CI did not support (S27, S51). In another example, one participant reported “We stopped using TRAVIS CI for this project because of random compile failures whenever the compiler ran out of RAM.” It appears that, given the baseline rate of more answers from abandoners, switchers are overrepresented regarding complaints about build speed.

In our model, *last build duration* is associated with a *decreased* chance of abandoning or switching CI. For every 1% increase in *last build duration*, the chance of switching and abandoning decreases by 57% and 67%, respectively, and this factor explains the highest fraction of variance in both models relative to the other non-control variables, with a large and medium effect size, respectively ( $\eta^2 = 0.21, 0.14$ ). However, build length itself does not indicate whether it is perceived as too long, and long builds could actually indicate that CI performs valuable analyses for the project.









We separately model *has long builds*, because we suspect non-linearity in that moderate build times are acceptable but very long build times indicate a barrier: we thus classify projects as having very *long builds* if they have builds much longer than their peers of similar size (full operationalization details above in Sec. 5). Those with very long builds are 126% more likely to abandon, though this factor explains the least amount of variance in the model, not even reaching a small effect size ( $\eta^2 = 0.004$ ).

*In summary, we find strong evidence in the survey that people abandon and switch because of long builds, in line with past literature, but our model finds the opposite effect for the absolute duration of the last build.*

### 6.2 Unsupported Technologies ☹

28 respondents left TRAVIS CI because it did not support technology they needed, the most common reason cited in our survey, aligning with observations O3.c: multiple tool use [13, 13, 27, 52], O3.d: lack of language support [4], and partially with O3.a: general configuration challenges [2, 4, 18, 21, 27, 39, 58, 60]. 11 respondents reported poor support for the Windows ecosystem as a reason for leaving TRAVIS CI (S4, S5, S32, S34, S40, S67, S96, S98, A71, A79, A129), including the .NET framework (S4, S34, S40, S67, S98, A129), C# language (S4, S98,

**Table 4: Results Summary.** A $\uparrow$ : Abandoning CI is statistically significantly more likely; A $\downarrow$ : Abandoning CI – less likely; S $\uparrow$ : Switching CI – more likely; S $\downarrow$ : Switching CI – less likely.

Theme	Active CI Users	TRAVIS CI Leavers	
	Literature Observation(s)	Support from Survey	Modeling Results
 <b>Unsupported tech.</b>	<b>O3.d:</b> Lack of support for a project’s language makes CI use difficult [4]. <b>O3.c:</b> Connecting developer tools to CI is challenging [13, 13, 27, 52]. <b>O3.a:</b> CI can be hard to configure [2, 4, 18, 21, 27, 39, 58, 60].	11 respondents left TRAVIS CI because it did not support the programming language they need (S4, S5, S32, S34, S40, S67, S96, S98, A71, A79, A129). 9 left TRAVIS CI because of poor Docker support (S8, S19, S20, S31, S58, S60, S89, S121, A100), and 8 needed other tools (S31, S57, S58, S75, S83, S97, S116, S125).	Project’s <i>language is supported</i> : A $\downarrow$ , S $\downarrow$ <b>Needs Docker: A<math>\uparrow</math> and S<math>\uparrow</math>, for new users only</b>
 <b>Long build times</b>	<b>O4.a:</b> Long builds frustrate users or make CI impractical [16, 27, 35, 37, 39, 40, 60]. <b>O4.b:</b> Long builds lead to pull request latency [82, 87]. <b>O4.c:</b> Long builds lead to an error prone setup [2, 4, 13, 52, 60]. <b>O4.d:</b> Data intensive projects make CI expensive. [2, 58, 59].	15 left TRAVIS CI because they found the build or queue times too long (S35, S4, S59, S75, A80, S99, S125, S128, S26, S27, S5, S51, S62, A100, A104). 5 other left TRAVIS CI because of RAM, space, or build time limits (S26, S81, A14, A23, A79).	<i>Has long builds</i> : A $\uparrow$ . <i>Longer last build duration</i> : A $\downarrow$ , S $\downarrow$
 <b>CI consistency</b>	N/A, see Sec. 6.3	17 left TRAVIS CI to achieve CI consistency across projects, or carry poor TRAVIS CI experiences to new projects (S4, S5, S8, S9, S31, S38, S59, S60, S63, S69, S74, S88, S93, S124, S126, S127, A17).	<i>Exposure to leavers</i> : <b>A<math>\uparrow</math>, S<math>\uparrow</math></b>
 <b>Lack of tests</b>	<b>O5.a:</b> Projects with few/no tests benefit less from CI. [28, 39, 40, 52, 64] <b>O2.f:</b> Projects with existing test culture benefit less from CI [28].	14 left TRAVIS CI because they lacked (sufficient) tests (S44, S91, A6, A18, A25, A50, A66, A68, A70, A86, A101, A102, A111, A115).	<i>Lower num. tests</i> : A $\uparrow$
 <b>Infrequent changes</b>	<b>O2.e:</b> Projects which change infrequently are less likely to need to use CI [28].	6 left TRAVIS CI because the project was inactive or had very low activity (S77, A48, A53, A82, A103, A118).	<i>Low activity in past 6 months</i> : A $\downarrow$ , S $\downarrow$
 <b>Poor user exp.</b>	<b>O3.b:</b> CI configuration files can be confusing, leading some to prefer a GUI [18, 27].	9 reported that UI concerns, poor documentation or community support influenced their decision to leave TRAVIS CI (S4, S26, S35, S38, S51, S62, S69, A80, S83).	N/A, see Sec. 6.6
 <b>Company support</b>	<b>O2.n:</b> Company supported projects are reluctant to use a public CI service [27].	6 left TRAVIS CI due to security concerns related to the proprietary nature of their test environment (S5, S15, S41, S57, S58, S69, S116, A64).	<i>Contributors with commercial users</i> : A $\downarrow$
 <b>Build failures</b>	<b>O1.a:</b> Troubleshooting build failures is difficult [27, 28, 39]. <b>O5.d:</b> Flaky tests or CIs compound this [13, 37, 39, 39, 52, 58, 58, 60].	4 found it difficult to troubleshoot a build failure (S58, S75, S83, A13). 6 left TRAVIS CI because of test or CI setup instability (S19, S59, S121, A14, A66, A108).	<i>Higher % of rebuilds</i> : <b>A<math>\uparrow</math>, S<math>\uparrow</math></b>

A71), and .NET package manager NuGet.org (S34, S67). For example, one respondent acknowledged that while TRAVIS CI supported Windows projects in a limited manner, they found this insufficient; e.g., “I stopped using TRAVIS CI at this point because I switched from Java to C# with .NET Core. I think I even tried to install .NET during part of the build steps but couldn’t get it to work very well.” Nine respondents specifically stated their need for Docker support (A100, S20, S31, S58, S89, S121, S19, S60, S8), all but one switching for this reason. For example, S89 responded “I switched to Wercker a long time ago. Wercker [is a] container based CI pipeline, which means it can execute any scripts with much more flexibility.” Finally, eight respondents reported that poor support for various other artifact and deployment services lead to their decision to switch TRAVIS CI (S31, S57, S58, S75, S83, S97, S116, S125). E.g., S58 explained: “Cloud-bees allowed us to publish artifacts on the Maven repo [...] for building “interim” versions [...] primarily for ourselves but for others too.”

We suspect that any technology without full support of the CI service could influence developers to switch. In addition, we suspect that developers realize a platform barrier early on: they either find a workaround or switch to a CI service without this barrier quickly.

Thus we expect that platform barrier will disproportionately affect projects new to a CI service.

Our model indicates that the lack of *language support* has a significant effect, associated with an increased risk of abandonment and switching, but *needing Docker* has no significant effects alone. However, our model shows that when interacted with *new users*, needing docker has significant, strong effects (illustrated in Figure 2c). *Needing Docker* increases *new users*’ chances of switching and abandoning by 942 % and 525 %, respectively, with a small effect size in each case ( $\eta^2 = 0.02, 0.01$ ), likely because the interaction is rare. The effect appears stronger for switching than abandoning, suggesting that many switch to a CI providing the necessary technology support rather than abandon CI altogether. This is in line with the fact that of the 28 respondents reporting unsupported technology, only four abandoned. (Note: Though we tried, we were unable to model a similar interaction for *language support*, because the VIF indicated excessive multicollinearity.) *Language support* decreases projects’ chances of switching and abandoning by 53 % and 51 %, respectively, with a small effect size in each model ( $\eta^2 = 0.03, 0.04$ ).

In summary, our survey and model confirm that lack of support for needed technology is associated with an increase risk of abandoning,

but more so switching to a CI platform with the necessary support, in line with previous research.

### 6.3 CI Consistency Across Projects 🏠

The third most common reason given for switching CI in our survey was that developers followed decisions in other projects to achieve consistency across projects. Even though not mentioned in previous literature, this reason had strong support from survey and model in our population.

Sixteen participants indicated that they switched CI services because they followed decisions in their other projects, and switched the project in question to maintain consistency (S4, S5, S8, S9, S31, S38, S59, S60, S63, S69, S74, S88, S93, S124, S126, S127), and one respondent abandoned CI for the same reason (A17). For example, one respondent said “I had some open source projects running in TravisCI and some in CircleCI. I just wanted to consolidate the project to one place and I’m sorry to say that at that time TravisCI lost the battle.”

Exposure to leavers increases projects’ chances of switching and abandoning by 183 % ( $\eta^2 = 0.04$ ) and 149 % ( $\eta^2 = 0.02$ ), respectively. Our model confirms a significant but small network effect, indicating that projects of contributors who have previously left TRAVIS CI are influenced, and thus more likely to switch or abandon CI.

In summary, our results indicate that the desire for CI consistency across projects plays an important role in a contributor’s choice of CI and that knowledge about CI flows across communities through people: projects with contributors who have participated in other projects that left TRAVIS CI in the past are more likely to switch and abandon.

### 6.4 Lack of Tests 🏠

Even though running tests is a key feature of CI, our survey indicates that many participants adopted TRAVIS CI with few or no tests for it to run. Aligning with O5.a: CI is less useful with few tests [28, 39, 40, 52, 64], but perhaps contradicting O2.f: an existing testing culture makes CI less useful [28], 14 participants cited the lack of tests, the lack of meaningful test coverage, or the difficulty of writing quality tests as a reason for leaving TRAVIS CI (S44, S91, A6, A18, A25, A50, A66, A68, A70, A86, A101, A102, A111, A115); e.g., A111 indicated “We abandoned Travis because [a certain API] requires an elaborate test setup. The goal [of adopting Travis] was to ‘force’ myself to add some real tests (to have green Travis badge again!) but this failed so far :)” Lack of tests was predominantly a barrier that lead to abandonment rather than switching (12 out of 14 participants).

In our model, a lower number of test files is significantly associated with an increased risk of abandonment, with no significant effect observed for switching. This is as expected: the problem of not having tests will not be solved by switching CI, thus no association is expected. For every 1% increase in the number of test files, the chance of abandoning decreases by 16 % ( $\eta^2 = 0.04$ ). To illustrate this effect, we plot the number of tests for per project In Figure 2b, where we see that abandoners have slightly fewer tests on average than switchers do or the control group.

In summary, our survey and model confirm past literature: CI is much less useful with few tests, associated with an increased chance of abandonment.

### 6.5 Infrequent Changes 🏠

Nine of our respondents reported that they left TRAVIS CI because their project had so little activity that they did not see the point of maintaining a CI platform and its associated overhead (A44, A48, A53, A82, A86, A102, A103, A118, S77), which aligns with findings in prior literature (O2.e) [28]. As one would expect, all but one of these participants abandoned CI on their project altogether.

However, these results do not generalize to our measure of low activity in the last 6 months: According to our model, projects with low activity are less likely to abandon and switch CI. Low activity decreases projects’ chances of switching and abandoning by 51 % and 73 %, having a small ( $\eta^2 = 0.02$ ) and medium effect size ( $\eta^2 = 0.08$ ) in the switching and abandoning models, respectively. This may be because across the GITHUB population at large, most projects with low activity perceive little CI overhead once configured or may not think to switch off CI even if it is not being used.

In summary, our survey supports low activity as a reason for abandoning in line with prior research, though our model does not.

### 6.6 Poor User Experience 🏠

Nine respondents mentioned aspects related to UI, documentation, and support when explaining why they left TRAVIS CI (S4, S26, S35, S38, S51, S62, S69, S83, A80). Four respondents mentioned that the UI was a factor in choosing their new CI (S26, S35, S38, S83), with these respondents preferring GUIs over configuration files, in accordance with O3.b [18, 27]. Other factors mentioned were the level of support (S26) and the quality of the documentation provided (S51). E.g., S26 expressed “Mostly I liked the fact [I can] conl all repository settings through their web interface, rather than including a ‘.travis.yml’ with a bunch of complicated options. Semaphore is very responsive to customers [implementing features based on feedback]. It feels like Semaphore really cares about us.” These responses align well with O3.b: configuration files are confusing, leading some to prefer a GUI. Difficulties largely motivate switching rather than abandonment.

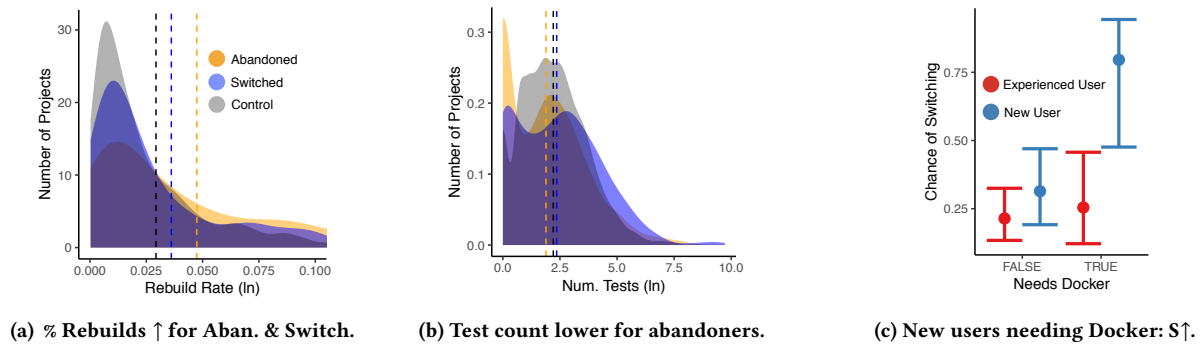
User experience is difficult to operationalize with archival data, hence we leave validation to future work in which other research methods such as user studies should be considered.

In summary, with the caveat that this is not a triangulated result, participants report that user experience concerns lead them to switch and abandon CI use, in line with previous findings.

### 6.7 Closed Source Concerns 🏠

Eight survey respondents mentioned that they left TRAVIS CI due to security or permissions concerns, often due to the proprietary, closed source nature of code associated with their open source project or test environment, even though the project itself was open source, aligning with with O2.n [27], (S5, S15, S41, S57, S58, A64, S69, S116). Specific reasons included the unwillingness to trust third parties with security certificates, the wish to run CI behind a firewall, the need to anonymize database information needed to run tests, and the need to run security tools unsupported by TRAVIS CI. E.g., S57 who switched to a self-hosted CI indicates “It is now easier for us to accommodate closed-source projects for our commercial customers without the security issues of source hosted by third parties such as TRAVIS CI.” Three respondents mentioned that they wanted the ability to run the same CI system on private repositories also, to be able to work with closed source code with the same tool, as





**Figure 2: Layered histograms for *Rebuild Rate* & *Number of Tests*, Dot and Whisker plot for *New User/Needs Docker* interaction.**

the free tier TRAVIS CI does not support private repositories (S41, S57, S69).

However, in our model, we find that projects with *commercial users* are less likely to abandon, with no significant observed effects on switching. *Commercial involvement* decreases projects’ chances of abandoning by 25%, but with a negligible effect size ( $\eta^2 = 0.008$ ).

*In summary, our survey responses confirm that commercially supported, closed source associated projects are reluctant to use a public CI service, in line with past literature, though our model does not.*

## 6.8 Troubleshooting Build Failures ❗

Four participants spoke about problems they ran into while debugging their projects on TRAVIS CI (S58, S75, S83, A13), corresponding to prior observations O1.a: troubleshooting build failures is difficult [27, 28, 39] and complicated by O5.d, concerning flaky tests and platforms [13, 27, 28, 37, 39, 39, 52, 58, 58, 60]. They mostly switched to competing CI services or local installations which allow them to debug their build as it was running (e.g., over SSH), such as S83: “Then [CircleCI] became my preference because they offered the ability to SSH in to debug, because the constant pushing/rebuilding for one print line got really old.” Furthermore, six respondents mentioned that they left due to issues with the CI service’s stability (S19, S59, A14, A66, A108, S121), e.g., S121 reports “We had a lot of issues getting Travis to build consistently and should’ve just removed it. [...] Sometimes the build was just hanging – not completing. Which lead to manually having to re-trigger it.” Specific complains included random and nondeterministic compilation failures (A14), a lack of build reproducibility (S59), and the testing environment failing before any tests could run (A108).

Our model indicates that *rebuilt*s of the same commit (an indicator of flakiness) associate significantly with both switching and abandoning CI. For every 1% increase in the percentage of *rebuilt*s, the chances of switching and abandoning increase by approximately 119% ( $\eta^2 = 0.02$ ) and 115% ( $\eta^2 = 0.02$ ), respectively. For illustration, we plot the distribution of the *rebuild rate* variable for abandoners, switchers, and control group projects in Figure 2a.

*In summary, survey and model confirm that troubleshooting CI build failures is difficult, especially with flaky tests or CI platforms. Projects with many rebuilt of the same commit are more likely to abandon and switch CI.*

## 7 DISCUSSION AND IMPLICATIONS

Our study replicated and confirmed several findings across two research methods, providing strong support that the certain discussed pain points are actually barriers to adopting CI or finding the right tool that affect broad user populations. We argue that both researcher and CI providers should explore solutions for those problems, possibly prioritized by severity of our findings.

### 7.1 Triangulated Results

We start with the triangulated results, suggesting ways to integrate existing research to improve CI processes and tools, where possible.

*Unsupported technology* was the most common issue in our sample. CI tool builders should enhance support for commonly used tools, especially Docker, because many found current support insufficient.<sup>7</sup> Likewise, as CI becomes more ubiquitous, CI providers may consider supporting a wider variety of languages, or make it easier for users to add support for niche languages. Perhaps more importantly, the CI community at large should consider highlighting this factor as a primary decision point when choosing a CI, because even though it is quick to check whether Docker or your language is supported, our survey shows that some only realize this will be an issue after spending the time to adopt TRAVIS CI.

*Lack of tests* is an important barrier to effective CI use, and getting people to write tests has been broadly recognized as difficult [56]. There has been work investigating the effects of providing incentives in an educational context for test writing [68], and also work showing that gamification using GITHUB badges can improve test coverage [72]. CI vendors or external services could consider integrating such mechanisms into the CI ecosystem (e.g., a “test suite degraded” badge to signal to a team that a developer’s commit did not include tests), and future research can find ways to further incentivize test writing. Survey respondents complained about the difficulty of writing GUI tests, so research on automating the creation of these tests [25], as well as automating test creation in general [17], can be integrated into newer versions of CI tools.

The strive for *CI consistency* across an organization (which is understandable from a desire to minimize context switches [46]) might encourage CI vendors in the current competitive market to support also niche requirements, as not meeting the technical requirements for a single project may have rippling effects for others.

<sup>7</sup><https://docs.travis-ci.com/user/docker/>

Although technical limitations for languages and tools, such as *containerization*, *.net*, and *GUI testing*, may be relevant only for few customers, they were frequently mentioned in our survey and are strong drivers for switching and abandoning for those projects affected, but may influence decisions on other projects as well. Where CI consistency is not possible or desirable, external tools could reduce CI context switching costs, for example providing compatible configuration files, interconnectivity, or shared dashboards.

Regarding *troubleshooting build failures*, it seems worth exploring new tools that learn from decades of debugging research [36, 54, 81, 84, 85] for the use in CI settings, where interactive debugging is less feasible. For example, debugging big-data analytics faces some similar challenges and recent advances [23] may provide some encouraging direction also for CI research. Research on usability and understandability of CI logs and how they could be presented better could provide significant benefits. Flaky tests seem to be a frequent and severe pain point, and work, such as identifying root causes of flaky tests [42], seems promising for improving CI.

## 7.2 Conflicting or Weak Results

Not all previously reported pain points gathered support from survey and model on our population. Indeed, some results were even seemingly contradictory and deserve further investigation.

Firstly, *company support*, and associated risks of exposing proprietary data or confidential information needed for testing otherwise open source code, was cited as a common reason for leaving the public TRAVIS CI system. CI tool builders may consider adding more features to partition open source components from associated proprietary or confidential code and data. However, our model across many projects suggests the opposite: commercially associated projects are more likely to abandon TRAVIS CI, with no effect observed for switchers. More work is thus needed, for example, to explore to what degree leavers may have specific requirements different from the general population of commercially associated open-source projects.

Secondly, *Long build times* were a commonly cited barrier to CI use. Test suite prioritization and selection could improve feedback times [62] and it already has some success in industry in a CI setting [16, 44, 45], e.g., prioritizing commits rather than tests [41]. Still, these techniques have not been explored much in public CI tools beyond *in vitro* studies [41]. TRAVIS CI recommends parallelizing test suite execution, but future iterations of CI tools should consider providing built-in support for test suite selection and prioritization. Our model found that having *very* long builds is associated with a very small but higher risk of abandonment, absolute *build duration* had a much stronger, negative association with leaving TRAVIS CI; future work should study developer perceptions of build length in more detail, to further investigate possible non-linearity.

Thirdly, *infrequent changes* were a commonly cited reason for CI abandonment, but our model showed the opposite: less active projects were *less* likely to abandon CI. We speculate that this may simply be because maintainers do not turn off CI on mostly (but not entirely) inactive projects; more research is needed given this conflicting result.

Finally, *user experience* issues were frequently mentioned in our survey, consistent with prior literature. Since we cannot operationalize these issues on trace data, we do not have confidence in the severity and pervasiveness of the issue.

## 7.3 Non Confirmed Observations

Finally, for many cited pain points in the literature we found no support in our survey or model. Beyond seven out of scope observations (see Sec. 5, e.g., O2.l: effects of CI on hiring), we found no mention in the survey at all for 11 observations. For example, no respondents complained about the difficulty of understanding the overall state of a project using CI (O1.b), nor a need for better notifications (O1.c); neither culture (O2.b) nor leadership buy-in problems (O2.c) were found in responses, nor were possibility of reputational damage (O2.j), concerns about CI being an unreachable standard (O2.i), or concerns about newcomer misunderstandings (O2.k). These may therefore amount to annoyances, but not things that outright lead people to switch or abandon CI altogether.

## 7.4 Implications for Open-Source Practitioners

Adopting and configuring CI can be time consuming. To improve the experience, we provide a concrete checklist of questions to ask, based on our research, that highlights the most common barriers to CI adoption before finding out problems the hard way after investing significant time in trying and configuring a CI system:

### **Does the CI platform support the languages and tools we use?**

Choosing a CI platform with native support for an existing setup will likely be much easier than hacking a CI platform to work with an unsupported language or tool. Consider at least operating system, programming languages, and container support, if needed. **Do our builds take a long time to run?** If builds take a long time to run with your current setup, they may take even longer on a public CI platform. Consider a local CI with dedicated hardware or a service with sufficient resources, or splitting up larger projects and testing them individually.

**Do other people in our organization currently use a different CI?** If so, ask them how they like it. If they abandoned it, ask them why. It is likely that their negative experiences may carry over to your the current project, and CI setup consistency across different projects in your organization may make things easier.

**Do we have tests?** CI can encourage people to write more tests, but CI might not be helpful if there are not already tests to run. Consider encouraging people to write tests before adopting CI.

**Do we interact with proprietary artifacts?** Even if your project is open source, interacting with proprietary artifacts in other parts of your organization makes it harder to have a consistent CI setup. Consider investing in a private CI setup for both open and closed source code to ensure consistency.

## 8 CONCLUSION

In this paper, we conducted a multiple conceptual multiple replication of observations derived from a literature review of 37 papers, using a mixed methods triangulation design comprised of 132 survey responses and 12 interviews from TRAVIS CI leavers and two logistic regression models on trace data from 6,239 TRAVIS CI projects. Comparing results from each method, we discuss which observations we were able to confirm, and also discuss what factors lead people to switch CI tools and which lead people to abandon CI altogether. We discuss how our work has actionable implications for CI research, CI tool builders, and CI users.

**Acknowledgements.** We thank our participants and the NSF (awards 1717415, 1318808, 1552944, 1717022).

## REFERENCES

- [1] Paul D Allison. 1999. *Multiple regression: A primer*. Pine Forge Press.
- [2] Brett K Beaulieu-Jones and Casey S Greene. 2017. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology* (2017).
- [3] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. In *International Conference on Software Engineering (ICSE)*. ACM, 12–23.
- [4] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In *International Conference on Mining Software Repositories (MSR)*. IEEE, 356–367.
- [5] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *International Conference on Mining Software Repositories (MSR)*. IEEE press, 447–450.
- [6] Michael Berkwitz and Thomas S Inui. [n. d.]. Making use of qualitative research techniques. *Journal of general internal medicine* ([n. d.]).
- [7] Martin Brandtner, Emanuel Giger, and Harald Gall. 2014. Supporting continuous integration by mashing-up software quality information. In *Conference on Software Maintenance, Reengineering and Reverse Engineering*. IEEE, 184–193.
- [8] Martin Brandtner, Sebastian C Müller, Philipp Leitner, and Harald C Gall. 2015. Sqa-profiles: Rule-based activity profiles for continuous integration environments. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 301–310.
- [9] Donald T Campbell and Donald W Fiske. 1959. Convergent and discriminant validation by the multitrait-multimethod matrix. *Psychological bulletin* (1959).
- [10] José Campos, Andrea Arcuri, Gordon Fraser, and Rui Abreu. 2014. Continuous test generation: enhancing continuous integration with automated test generation. In *International conference on Automated software engineering (ASE)*. ACM, 55–66.
- [11] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM.
- [12] Ricardo Colomo-Palacios, Eduardo Fernandes, Pedro Soto-Acosta, and Xabier Larrucea. 2018. A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management* (2018).
- [13] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. 2014. Challenges when adopting continuous integration: A case study. In *International Conference on Product-Focused Software Process Improvement*. Springer, 17–32.
- [14] Mary Dixon-Woods, Rachel L Shaw, Shona Agarwal, and Jonathan A Smith. 2004. The problem of appraising qualitative research. *BMJ Quality & Safety* (2004).
- [15] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. 2008. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*. Springer, 285–311.
- [16] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 235–245.
- [17] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Symposium on the Foundations of Software Engineering (FSE)*. ACM, 416–419.
- [18] Keheliya Gallaba and Shane McIntosh. 2018. Use and Misuse of Continuous Integration Features: An Empirical Study of Projects that (mis) use Travis CI. *Transactions on Software Engineering (TSE)* (2018).
- [19] Lisa M Given. 2008. *The Sage encyclopedia of qualitative research methods*. Sage Publications.
- [20] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: GitHub's data from a firehose. In *Working Conference on Mining Software Repositories (MSR)*. IEEE, 12–21.
- [21] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *International Conference on Software Engineering (ICSE)*. IEEE, 285–296.
- [22] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *International Conference on Software Engineering (ICSE)*. IEEE Press, 358–368.
- [23] Muhammad Ali Gulzar, Xueyuan Han, Matteo Interlandi, Shaghayegh Mardani, Sai Deep Tetali, Todd D Millstein, and Miryung Kim. 2016. Interactive Debugging for Big Data Analytics.. In *HotCloud*.
- [24] Yash Gupta, Yusaira Khan, Keheliya Gallaba, and Shane McIntosh. 2017. The impact of the adoption of continuous integration on developer attraction and retention. In *International Conference on Mining Software Repositories (MSR)*. IEEE, 491–494.
- [25] Leegeun Ha, Sungwon Kang, Jihyun Lee, and Younghun Han. 2018. Automatic Generation of GUI Test Inputs Using User Configurations. In *International Conference on Big Data, Cloud Computing, and Data Science Engineering*. Springer, 103–116.
- [26] Ralph Hertwig, Carola Farnelov, and Ulrich Hoffrage. 2003. Hindsight bias: How knowledge and heuristics affect our reconstruction of the past. *Memory* (2003).
- [27] Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. 2017. Trade-offs in continuous integration: assurance, security, and flexibility. In *Foundations of Software Engineering (FSE)*. ACM, 197–207.
- [28] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *International Conference on Automated Software Engineering (ASE)*. IEEE, 426–437.
- [29] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. Vol. 398. John Wiley & Sons.
- [30] Xin Huang, He Zhang, Xin Zhou, Muhammad Ali Babar, and Song Yang. 2018. Synthesizing qualitative research in software engineering: a critical review. In *International Conference on Software Engineering (ICSE)*. ACM, 1207–1218.
- [31] Md Rakibul Islam and Minhaz F Zibran. 2017. Insights into continuous integration build failures. In *International Conference on Mining Software Repositories (MSR)*. IEEE, 467–470.
- [32] Natalia Juristo and Omar S Gómez. 2012. Replication of software engineering experiments. In *Empirical software engineering and verification*. Springer.
- [33] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *International Conference on Mining Software Repositories (MSR)*. ACM, 92–101.
- [34] David Kavaler, Asher Trockman, Bogdan Vasilescu, and Vladimir Filkov. 2019. Tool Choice Matters: JavaScript Quality Assurance Tools and Usage Outcomes in GitHub Projects. In *International Conference on Software Engineering (ICSE)*. IEEE.
- [35] Eric Knauss, Miroslaw Staron, Wilhelm Meding, Ola Söder, Agneta Nilsson, and Magnus Castell. 2015. Supporting continuous integration by code-churn based test selection. In *International Workshop on Rapid Continuous Software Engineering*. IEEE Press, 19–25.
- [36] Andrew Ko and Brad Myers. 2008. Debugging reinvented. In *International Conference on Software Engineering (ICSE)*. IEEE, 301–310.
- [37] Adriaan Labuschagne, Laura Inozemtseva, and Reid Holmes. 2017. Measuring the cost of regression testing in practice: a study of Java projects using continuous integration. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 821–830.
- [38] Davy Landman, Alexander Serebrenik, and Jurgen J Vinju. 2017. Challenges for Static Analysis of Java Reflection-Literature Review and Empirical Study. In *International Conference on Software Engineering (ICSE)*. IEEE, 507–518.
- [39] Eero Laukkanen, Maria Paasivaara, and Teemu Arvonen. 2015. Stakeholder Perceptions of the Adoption of Continuous Integration—A Case Study. In *Agile Conference (AGILE)*. IEEE, 11–20.
- [40] Marko Leppanen, Simo Makinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V Mantyla, and Tomi Mannisto. 2015. The highways and country roads to continuous deployment. *IEEE Software* 2 (2015), 64–72.
- [41] Jingjing Liang, Sebastian Elbaum, and Gregg Rothermel. 2018. Redefining prioritization: Continuous prioritization for continuous integration. In *International Conference on Software Engineering (ICSE)*. IEEE.
- [42] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An empirical analysis of flaky tests. In *International Symposium on Foundations of Software Engineering (FSE)*. ACM, 643–653.
- [43] Marco Manglaviti, Eduardo Coronado-Montoya, Keheliya Gallaba, and Shane McIntosh. 2017. An empirical study of the personnel overhead of continuous integration. In *International Conference on Mining Software Repositories (MSR)*. IEEE Press, 471–474.
- [44] Dusica Marijan, Arnaud Gotlieb, and Sagar Sen. 2013. Test case prioritization for continuous regression testing: An industrial case study. In *International Conference on Software Maintenance (ICSM)*. IEEE.
- [45] Atif Memon, Zebao Gao, Bao Nguyen, Sanjeev Dhandu, Eric Nickell, Rob Siemborski, and John Micco. 2017. Taming Google-scale continuous testing. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*.
- [46] André N Meyer, Thomas Fritz, Gail C Murphy, and Thomas Zimmermann. 2014. Software developers' perceptions of productivity. In *International Symposium on Foundations of Software Engineering (FSE)*. ACM, 19–29.
- [47] Jeremy Miles and Mark Shevlin. 2001. *Applying regression and correlation: A guide for students and researchers*. Sage.
- [48] Craig S Miller. 2011. Item sampling for information architecture. In *Conference on Human Factors in Computing Systems (CHI)*. ACM, 2211–2214.
- [49] Audris Mockus. 2008. Missing data in software engineering. In *Guide to advanced empirical software engineering*. Springer, 185–200.
- [50] Gareth Morgan. 1997. *Images of organization*. SAGE Publications.
- [51] Anton J Nederhof. 1985. Methods of coping with social desirability bias: A review. *European journal of social psychology* (1985).
- [52] 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 *Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 392–399.

- [53] Klérissou VR Paixão, Cricia Z Felício, Fernanda M Delfim, and Marcelo de A Maia. 2017. On the interplay between non-functional requirements and builds on continuous integration. In *International Conference on Mining Software Repositories (MSR)*. IEEE Press, 479–482.
- [54] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 199–209.
- [55] Jagdish K Patel, CH Kapadia, Donald Bruce Owen, and JK Patel. 1976. *Handbook of statistical distributions*. Technical Report. M. Dekker New York.
- [56] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. 2013. Creating a shared understanding of testing culture on a social coding site. In *International Conference on Software Engineering (ICSE)*. IEEE Press, 112–121.
- [57] Gustavo Pinto, Fernando Castor, Rodrigo Bonifacio, and Marcel Rebouças. 2018. Work Practices and Challenges in Continuous Integration: A Survey with Travis CI Users. In *Software - Practice And Experience*. Wiley.
- [58] Gustavo Pinto, Marcel Rebouças, and Fernando Castor. 2017. Inadequate testing, time pressure, and (over) confidence: a tale of continuous integration users. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE Press, 74–77.
- [59] Denis Polkhovskiy. 2016. Comparison between continuous integration tools. (2016).
- [60] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. 2017. An empirical analysis of build failures in the continuous integration workflows of Java-based open-source software. In *International Conference on Mining Software Repositories (MSR)*. IEEE Press, 345–355.
- [61] Peter Reason and Hilary Bradbury. 2001. *Handbook of action research: Participative inquiry and practice*. Sage.
- [62] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test case prioritization: An empirical study. In *International Conference on Software Maintenance (ICSM)*. IEEE, 179–188.
- [63] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. 2016. Continuous deployment at Facebook and OANDA. In *International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 21–30.
- [64] Gerald Schermann, Jürgen Cito, Philipp Leitner, and Harald C Gall. 2016. Towards quality gates in continuous delivery and deployment. In *International Conference on Program Comprehension (ICPC)*. IEEE, 1–4.
- [65] Stefan Schmidt. [n. d.]. Shall we really do it again? The powerful concept of replication is neglected in the social sciences. *Review of General Psychology* ([n. d.]).
- [66] W Richard Scott. 1981. *Organizations: rational, natural, and open systems*. NJ: Prentice Hall, Englewood Cliff.
- [67] Reginald G Smart. 1966. Subject selection bias in psychological research. *Canadian Psychologist/Psychologie canadienne* (1966).
- [68] Jaime Spacco and William Pugh. 2006. Helping students appreciate test-driven development (TDD). In *Companion to the Symposium on Object-oriented Programming Systems, Languages, And Applications (OOPSLA)*. ACM, 907–913.
- [69] Daniel Ståhl and Jan Bosch. [n. d.]. Industry application of continuous integration modeling: a multiple-case study. In *International Conference on Software Engineering Companion (ICSE-C)*.
- [70] Kate Stewart. 1998. An exploration of customer exit in retail banking. *International Journal of Bank Marketing* 16, 1 (1998), 6–14.
- [71] Kristín Fjólá Tómasdóttir, Mauricio Aniche, and Arie van Deursen. 2017. Why and how JavaScript developers use linters. In *International Conference on Automated Software Engineering (ASE)*. IEEE, 578–589.
- [72] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *International Conference on Software Engineering (ICSE)*. ACM, 511–522.
- [73] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. Determinants of Sustained Activity in Open-Source Projects: A Case Study of the PyPI Ecosystem. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 392–399.
- [74] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. 2016. The sky is not the limit: multi-tasking across GitHub projects. In *International Conference on Software Engineering (ICSE)*. IEEE, 994–1005.
- [75] Bogdan Vasilescu, Stef Van Schuylenburg, Jules Wulms, Alexander Serebrenik, and Mark GJ van den Brand. 2014. Continuous integration in a social-coding world: Empirical evidence from GitHub. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 401–405.
- [76] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 805–816.
- [77] Carmine Vassallo, Fabio Palomba, Alberto Bacchelli, and Harald C Gall. 2018. Continuous code quality: are we (really) doing that?. In *International Conference on Automated Software Engineering (ASE)*. ACM.
- [78] David Gray Widder, Michael Hilton, Christian Kästner, and Bogdan Vasilescu. 2018. I'm Leaving You, Travis: A Continuous Integration Breakup Story. In *International Conference on Mining Software Repositories (MSR)*.
- [79] Timothy Williams, Jessie Schutt-Aine, and Yvette Cuca. 2000. Measuring family planning service quality through client satisfaction exit interviews. *International Family Planning Perspectives* (2000), 63–71.
- [80] Jeffrey M Wooldridge. 2015. *Introductory econometrics: A modern approach*. Nelson Education.
- [81] Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. 2014. CrashLocator: locating crashing faults based on crash stacks. In *International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 204–214.
- [82] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for it: Determinants of pull request evaluation latency on GitHub. In *International Conference on Mining software repositories (MSR)*. IEEE, 367–371.
- [83] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How open source projects use static code analysis tools in continuous integration pipelines. In *International Conference on Mining Software Repositories (MSR)*. IEEE, 334–344.
- [84] Andreas Zeller. 1999. SubjectYesterday, my program worked. Today, it does not. Why?. In *Joint Meeting on the Foundations of Software Engineering (ESEC/FSE)*.
- [85] Xiangyu Zhang, Rajiv Gupta, and Youtao Zhang. 2003. Precise dynamic slicing algorithms. In *International Conference On Software Engineering (ICSE)*. IEEE Computer Society, 319–329.
- [86] Yang Zhang, Bogdan Vasilescu, Huaimin Wang, and Vladimir Filkov. 2018. One Size Does Not Fit All: An Empirical Study of Containerized Continuous Deployment Workflows. In *Joint Meeting on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 295–306.
- [87] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In *International Conference on Automated Software Engineering (ASE)*. IEEE, 60–71.