I'm Leaving You, Travis: A Continuous Integration Breakup Story

David Gray Widder, Michael Hilton, Christian Kästner, Bogdan Vasilescu
Carnegie Mellon University, USA

ABSTRACT

Continuous Integration (CI) services, which can automatically build, test, and deploy software projects, are an invaluable asset in distributed teams, increasing productivity and helping to maintain code quality. Prior work has shown that CI pipelines can be sophisticated, and choosing and configuring a CI system involves tradeoffs. As CI technology matures, new CI tool offerings arise to meet the distinct wants and needs of software teams, as they negotiate a path through these tradeoffs, depending on their context. In this paper, we begin to uncover these nuances, and tell the story of open-source projects falling out of love with Travis, the earliest and most popular cloud-based CI system. Using logistic regression, we quantify the effects that open-source community factors and project technical factors have on the rate of Travis abandonment. We find that increased build complexity reduces the chances of abandonment, that larger projects abandon at higher rates, and that a project's dominant language has significant but varying effects. Finally, we find the surprising result that metrics of configuration attempts and knowledge dispersion in the project do not affect the rate of abandonment.

1 INTRODUCTION

Continuous Integration (CI) systems automate the compilation, building, testing, and deployment of software at a rapid pace [15]. CI is considered a software development best practice, and is known to improve productivity in software teams and help maintain code quality [3, 15, 26, 31]. However, rarely is a best practice truly *universal* and divorced from the context where it is applied. This intuitive concept is best described by *Contingency Theory* [20] in a wider organizational context: an organization's structures and processes should be compatible with the context in which it operates.

CI pipelines are no exception. Configuring CI is a complex process, with numerous tradeoffs [14] and side effects on other software development practices [32]. For example, Hilton *et al.* [14] found, after interviewing 16 industrial developers, that implementing CI requires negotiating tradeoffs between 1) test suite execution speed and certainty in the code's correctness, 2) ease of access and information security, and 3) the desire for many configuration options balanced against the desire for simplicity. Given all these choices, it's natural for different projects to have different needs, as well as for a project's CI needs to change over time, as developers navigate these tradeoffs [2, 27]. Simply stated, one size does not fit all.

As a realization of the diversity of CI needs, the marketplace of available CI tools in open-source software (OSS) is booming. While not long ago Jenkins and Travis CI were the only broadly used client-side and cloud-based CI services, respectively, there are now 17 CI services that integrate with GitHub directly,[1] and online discussions are explicit about the need for tailored CI solutions.[2]

To better support developers looking for bespoke CI solutions as well as designers and builders of CI tools, in this paper we begin to uncover the factors that inform the choice of CI implementations by OSS developers on GitHub. Specifically, we investigate 1,819 OSS projects moving away from Travis CI, the dominant cloud-based CI service on GitHub [15], and try to uncover, using multiple regression modeling, hints in their repositories that might explain and contextualize this transition.

The phenomenon of Travis CI abandonment, we argue, is a particularly attractive setting to study limitations of generic CI solutions. First, by virtue of its popularity, there is a wealth of public data available for GitHub projects using (and abandoning) Travis. Second, abandoning Travis is a more significant event in the life of a software project than its adoption: it is a choice to either abandon CI altogether, going against a best practice, or abandon just Travis CI, the most popular and trusted CI tool [15], in favor of a newer and presumably less recognized replacement. For this to occur, we assert there must be significant 'push' factors, which cause projects to abandon Travis or CI altogether, or overriding 'pull' contextual factors, which draw a project from Travis to a new CI service. Additionally, as the cost of setting up CI (including Travis) is non negligible [11, 24], abandoning suggests that the push or pull factors must be strong enough to overcome the initial investment. Therefore, investigating Travis abandonment is important to fully understand CI use, to understand how tool choices change with recent addition of alternatives, and as a more general case study of the factors which regulate community change.

Specifically, our results have the following highlights:

- We find that projects with more complex Travis configurations tend to be less likely to abandon Travis, providing support for the applicability of Contingency Theory in this context.
- We find that projects with more commits tend to be more likely to abandon Travis, suggesting that larger projects may outgrow Travis.
- Surprisingly, we find that metrics concerning number of attempts to configure Travis and the extent to which Travis configuration knowledge pervades the project community are not predictive of Travis abandonment.

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196422

[1] https://github.com/works-with
[2] E.g., https://github.com/blog/2463-github-welcomes-all-ci-tools

- We find that projects built with C# and other special purpose languages tend to be more likely to abandon Travis, perhaps due to a contextual mismatch.

2 DEVELOPMENT OF RESEARCH QUESTIONS

CI is a well-studied concept. Prior work has found that CI can help teams scale, increasing the numbers of developers and the size of the code base, without affecting software quality [26, 31]. Other work suggests that CI needs change over time as projects navigate different tradeoffs [27]. Retaining contributors is important for sustaining the life of distributed software projects [10], and CI usage has been shown to correlate with popularity: popular projects with more developers tend to use CI at higher rates [10]. The use of CI in a project, *e.g.*, as indicated by repository badges [29], is also a signal of adherence to best practices. CI itself can provide signals of differential interest to different stakeholders, and work is being done to unify signals from disparate sources and present only the signals which will be relevant to a given stakeholder [6, 7].

However, adoption of CI is not without barriers, many of them social or cultural, relating to incompatibility between the goals of CI and those of developers, and social processes within an organization, underscoring that CI may not be for everyone [11, 19, 24]. The problems of selecting appropriate test suites (*e.g.*, to balance testing speed and certainty), and automatically generating test suites, are also ones of recent study [9, 13, 18]. These problems may lead some organizations to only partially adopt CI into their workflow [21].

In this paper, starting from 11 interviews with developers involved in configuring CI in their projects, carried out during prior work [14], we built a "design space" of the facets of variability involved in configuring CI, and the tradeoffs which are made when choosing how to set up CI systems. From this, it was apparent that abandonment is likely to be a complex process, influenced by many factors. We group these factors into two distinct groups: technical attributes of the project itself and those pertaining to community structure. The goal of this study is to estimate the (relative) effects of these different factors on the likelihood of abandoning Travis.

Testing is arguably the most important stage in a CI pipeline [31], and Travis can be more useful in a test-heavy environment. Still, the importance of tests likely depends on a project's application domain, which is often reflected in the choice of programming language. HTML projects are likely to be web-related, R projects are likely to be scientific or statistical applications, and Objective C projects are likely to be mobile apps. Moreover, building and testing a project can be a multi-stage process. The better the fit between Travis' capabilities and a project's build needs, one can argue, the more of these stages it should be able to automate. In short, given that technical attributes of a project can indicate how much value the project gains from using CI, we ask: ***RQ1****: What effects do technical factors have on the likelihood of Travis abandonment?*

Community culture, values, and structure can have large and pervasive effects on that community's choices and processes. For example, projects with a culture that values stability are less likely to make changes which might affect users, at the cost of implementing new features. Other projects negotiate this tradeoff differently, valuing adding new features over stability [5]. As another example, there is evidence that some projects choose to use older versions of APIs, perhaps on the assumption that they are more stable, or that refactoring their project to work with new APIs may introduce defects [22]. Also, it is reasonable to expect that DevOps culture, of which CI is a central pillar, will not fit every community. In this case, communities may have decided to try out CI only to find that it clashed with their modus operandi. For example, a 2015 study identified cultural factors which may bar teams from embracing DevOps culture, such as a lack of agreement among team members as to the goals and values of a project, resistance to changing old habits, and the perception that embracing DevOps involves more work [28]. Additionally, they found that lack of developer interest in the "other side" of a Developer-Operations divide may lead to a rejection of DevOps culture. The important and varied role that community factors can have in the adoption and abandonment of a tool, and the culture in which its use can be fully realized, lead us to ask: ***RQ2****: What effect do community contextual factors have on the likelihood of Travis abandonment?*

3 METHODS

3.1 Dataset and Preprocessing

Starting from a data set of GitHub projects using (or having used) Travis CI, collected during prior work [32] *circa* March 2017, we identified 38,214 projects that had disabled Travis, using the binary `is active` flag returned by the Travis API.[3] Of these, 95% had also ceased development activity shortly after disabling Travis, *i.e.*, had no commits beyond 30 days after Travis abandonment;[4] we subsequently filtered these out, as the abandonment event cannot be disentangled from their termination of activity. We further filtered out infrequent programming languages (to allow for enough variance in the regression model). The resulting sample contains 1,819 projects distributed over 14 languages: 63 C#, 87 Puppet, 114 Shell, 134 CSS, 146 HTML, 155 Objective-C, 189 C, 234 C++, 283 Go, 651 Java, 868 Python, 947 PHP, 1,206 Ruby, and 2,199 JavaScript.

Next, we compiled a control group, down-sampling the much larger group of non-abandoning projects using nearest-neighbor propensity score matching [8] on the Travis adoption date (*i.e.*, the date of the first build); we sampled three matching non-abandoning projects for every abandoning one, using the `matchIt` R package. This ensures that all control-group projects started using Travis at (approximately) the same time as their treatment-group counterparts, therefore all had the same number of CI options available at the time of Travis adoption and, thus, controls for global environment effects. Our final sample of 7,276 projects is a merger of these control and treatment groups; Table 1 presents summary statistics.

3.2 Measures

The outcome measure is ***Travis CI abandonment***, as indicated by builds being switched off on the `travis-ci.org` dashboard; this is a manual action which must be undertaken by someone with administrator privileges to the repository. We further compute the following community (c) and technical (t) factors:

Project age (c/t): Project age in days. Older projects may be more entrenched in their practices, and thus less likely to abandon Travis.

[3] This guarantees that a project abandoned Travis on all branches, more accurately than branch-dependent signals, *e.g.*, the absence of a `travis.yml` configuration file.
[4] We extract commit data by cloning the repositories locally and parsing their `git` logs.

Table 1: Summary Statistics for Predictors

Predictor, Abandoned		Min	1Q	Med	Mean	3Q	Max
Project age (days)	T	1,079	1,377	1,630	1,682	1,946	2,932
	F	1,079	1,351	1,603	1,668	1,910	3,000
Contributors	T	1	2	5	13	13	143
	F	1	2	3	8	8	142
.yml contributors	T	1	1	1	2	2	11
	F	1	1	1	2	2	12
.yml commits	T	1	2	4	8	9	215
	F	1	2	4	7	8	217
Pull requests	T	0	0	0	18	8	644
	F	0	0	3	28	18	810
Commits	T	1	64	163	456	448	54,690
	F	2	27	66	218	180	24,012
Build duration (sec)	T	2	19	73	197	200	2,854
	F	2	31	126	267	301	2,998
Build jobs	T	1	1	1	2	2	34
	F	1	1	2	3	4	66

Contributors (c): How many distinct contributors have committed to this project? It may be harder to reach a consensus to modify development practices in a larger community.
.travis.yml contributors (c): The fewer people are involved in configuring Travis, the easier it may be to reach consensus to change.
.travis.yml commits (c): More investment in configuring Travis, therefore in using Travis, may suggest a decreased likelihood of abandonment. Alternatively, more commits may indicate possible frustration at trying to get Travis to perform as desired, thus increasing the likelihood of abandonment.
Pull request count (c): Travis is often used as a quality control gate for outside contributions, typically submitted as pull requests.
Commit count (c/t): As a proxy for project size.
Build duration (t): Duration of the most recent build, in seconds. Long build times (either from high build complexity, or heavy `travis-ci.org` server load) may push people off Travis, as witnessed qualitatively in log files, but may also indicate more substantive Travis usage and thus more commitment to Travis.
Build jobs (t): The number of jobs spawned per build is an indication of build complexity, the amount of investment in the Travis setup, and possibly related to the extent to which teams are able to configure Travis to suit their specific needs. Projects running more Travis jobs may have been able to tailor Travis to their context better, and may thus be less likely to abandon it.
Language (t): The dominant project language, as identified by GitHub. Languages with higher domain specificity may be less well suited to Travis, constituting a push factor, or especially well suited to other CI systems, constituting a pull factor.

3.3 Binomial Logistic Regression Modeling

Binomial Logistic Regression models estimate the likelihood of a binary outcome given a set of predictors. In our case, we have a sample of GitHub projects, a quarter of which abandoned Travis, matched on the Travis adoption date to control for environmental effects. Logistic regression allows us to *explain* the likelihood of the dependent binary event ***Travis CI abandonment***, as a function of our contextual measures which serve as the predictors, and to estimate the size of the effect of each variable on increased likelihood of abandonment, *while holding the other variables fixed.*

Table 2: Travis CI Abandonment Logit Model. McFadden $R^2 = 0.215$

Predictor	Coeffs (Errors)	Deviance
(Intercept)	3.33 (1.06)**	
log(Project age)	−0.68 (0.15)***	21.56***
log(Build duration)	−0.43 (0.03)***	226.27***
log(Commits)	0.62 (0.04)***	334.69***
log(Contributors)	0.34 (0.05)***	58.86***
log(Build jobs)	−0.33 (0.06)***	27.92***
log(Pull requests + 0.5)	−0.43 (0.02)***	423.17***
log(.yml commits)	0.02 (0.04)	0.30
log(.yml contributors)	0.02 (0.09)	0.07
Langauge: C vs Mean	−1.27 (0.21)***	
Langauge: C# vs Mean	0.81 (0.27)**	
Langauge: C++ vs Mean	−0.41 (0.17)*	
Langauge: CSS vs Mean	0.11 (0.19)	
Langauge: Go vs Mean	−0.03 (0.14)	
Langauge: HTML vs Mean	0.23 (0.20)	
Langauge: Java vs Mean	−0.22 (0.10)*	260.80***
Langauge: JavaScript vs Mean	−0.77 (0.08)***	
Langauge: Objective-C vs Mean	−0.26 (0.19)	
Langauge: PHP vs Mean	0.01 (0.10)	
Langauge: Puppet vs Mean	2.65 (0.25)***	
Langauge: Python vs Mean	−0.72 (0.10)***	
Langauge: Ruby vs Mean	−0.26 (0.09)**	

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

We built a binomial logistic regression model with ***Travis CI abandonment*** as response, using all the factors above as predictors. In preparation, we filtered out the top 1% of highly-skewed measures as potential high-leverage points, to increase model robustness [25, 30], and we log-transformed variables as needed, to stabilize variance and reduce heteroscedasticity [16]. We further performed multicollinearity analysis, checking if the *Variance Inflation Factor* remained below 3 [1]. The reference for categorical *language* data was set to the overall mean (*i.e.*, we used deviation coding); languages with positive coefficients are more likely to abandon Travis than the mean.

For each predictor, the model gives four useful pieces of information: whether the effect of the predictor on the outcome is statistically significant at 0.05 level or below, as indicated by the p-value; whether the effect is positive or negative, represented by the coefficient's sign; the strength of the effect, represented by the magnitude of the coefficient; and the share of the total variance explained by that predictor shown by an ANOVA type-2 analysis, revealing the relative importance of each predictor.

The model fits the data well: the McFadden ρ^2 value, a psuedo R^2 value for assessing goodness of fit in generalized linear models [23] (`pscl` package in R), is 0.215; the Area Under the Sensitivity/Specificity Curve (AUC) (`pROC` package) is 0.81.

4 RESULTS AND DISCUSSION

We now answer our research questions and discuss our results.
RQ1*: What effect do technical factors have on the likelihood of Travis CI Abandonment?* The dominant technical factors which explain

Travis abandonment are ***Build duration*** and ***Language***, with some deviance explained by the number of ***Build jobs*** run on Travis. Projects written in C#, mainly used for building Windows applications, are more likely to abandon Travis than average; this is unsurprising since Travis provides MacOS and Linux testing platforms, but does not provide one for Windows. Languages used in contexts in which testing may not make much sense, such as the software configuration management language Puppet, are also more likely to abandon Travis than average. Travis CI itself is built in Ruby, so naturally it has done a good job of catering to the Ruby language: the relative likelihood that Ruby projects abandon Travis is lower than average. Java is easily unit tested, and perhaps this translates to our observed lower chances of Travis abandonment than average.

Projects with a longer ***Build duration*** are less likely to abandon Travis. We speculate that because ***Build duration*** can indicate build complexity, projects with more complex builds are better able to adapt Travis to fit their specific context; this effect is supported in smaller part by Job Count. Overall this effect aligns with Contingency Theory: projects would not unnecessarily configure their build system to work in an arbitrary and complex way, but they might configure their build system in a complex way if the complexity reflected and fit well with their context; thus, higher complexity can show that Travis fits their context better, which may explain the reduced chances of abandonment.

RQ2*: What effect do community factors have on the rate of Travis CI Abandonment?* We will begin by discussing the community factors which, surprisingly, had little or no effect on the likelihood of Travis abandonment. One might assume that projects with more ***Contributors*** might be less likely to abandon Travis, *e.g.*, for fear of upsetting their large contributor base, but this effect does not play out in our model. Additionally, one might assume that the number of ***.yml commits*** to the Travis configuration file might affect the likelihood of Travis abandonment, whether negatively because of the time investment spent configuring Travis, or positively because of frustration over configuring Travis adequately, but the model does not suggest any such effects. Finally, one might assume that the extent to which Travis configuration knowledge is widespread within a project's community, as measured by the number of distinct ***.yml contributors***, would have some effect on the rates of Travis abandonment, but our model does not offer any indication of this.

The dominant community factors which explain Travis abandonment are the number of ***Commits***, and the number of ***Pull requests***. Projects with more ***Pull requests*** tend to be less likely to abandon Travis, other variables held constant; we speculate that this is because they are getting value out of Travis by using it to help evaluate external pull requests. Projects with more ***Commits*** are more likely to abandon Travis, suggesting that they may either become too large or unwieldy to work well with Travis, or that they have become large enough to justify a more customizable or tailored solution that better fits the context of their project.

5 THREATS TO VALIDITY

A single research method can never capture all the nuance involved in complex phenomena, and the blind spots of quantitative research on software repositories are well understood [4, 17]. While our model explains a sizeable chunk of the variance in Travis abandonment, it is quite likely that we neglected to include significant covariates of Travis abandonment. It is also possible that some of our measures don't accurately capture the concepts they try to operationalize, especially in the cases where the model does not suggest any significant effects. We also note that the randomness inherent in the propensity-score matching process may create a small amount of variability in the effects for languages represented by relatively few projects. Finally, we cannot make causal claims using our model, because it is possible that the factors we measured are instead correlated with an underlying and unknown true cause we do not account for. To enable replications and extensions, we make our data publicly available.[5]

6 CONCLUSIONS AND IMPLICATIONS

We have shown that a model with relatively simple predictors explains Travis abandonment with good fit.

Our results have three main implications on research. Firstly, our work motivates more research on understanding CI Abandonment, a previously unstudied and unquantified phenomenon. Secondly, our work suggests that more qualitative research is needed on how knowledge of a project's CI practices propagates through a community, since the measures we used to operationalize this concept did not have the expected effects. Finally, our work motivates more research on understanding how context affects the choice of CI, on which there have been little large scale quantitative studies.

Our work also has implications for practitioners. Firstly, users of Travis are encouraged to look elsewhere if Travis is not able to accommodate the complexity appropriate to their context. Secondly, projects using Travis should be wary of outgrowing their CI setup as they become larger. Finally, designers of Travis and similar CI services should pay attention to the context of use, and where there are particular unmet needs.

Future Work. Although encouraging, our results should be considered preliminary. Future work should focus on the causal mechanisms behind making the decision to abandon Travis. One direction could be to collect qualitative data from projects abandoning Travis, using a mixed-methods design [12]. Another direction could be to study what happens *after* projects abandon Travis. How many projects abandon CI completely? Alternatively, how many switch to a new CI system? Can contextual factors predict which new system they migrate to? Finally, future work should also consider the productivity and code quality implications of abandoning Travis or CI altogether. For example, does the increase in productivity associated with CI adoption, reported by prior work, come with an equivalent but opposite decrease in productivity after CI abandonment?

Acknowledgements. The authors would like to thank Momin Malik for his help with early model design, and Jim Herbsleb for his feedback on early drafts of this paper. Kästner's work has been supported in part by the NSF (awards 1318808, 1552944, and 1717022), and AFRL and DARPA (FA8750-16-2-0042). Widder and Vasilescu gratefully acknowledge NSF support (award 1717415).

[5] https://github.com/CMUSTRUDEL/travis-abandonment-data

REFERENCES

[1] Paul D Allison. 1999. *Multiple regression: A primer.* Pine Forge Press.

[2] Abigail Atchison, Christina Berardi, Natalie Best, Elizabeth Stevens, and Erik Linstead. 2017. A time series analysis of TravisTorrent builds: to everything there is a season. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, 463–466.

[3] M. Beller, G. Gousios, and A. Zaidman. 2017. Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub. In *Proc. International Conference on Mining Software Repositories (MSR)*. 356–367. DOI: http://dx.doi.org/10.1109/MSR.2017.62

[4] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. 2009. The promises and perils of mining git. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, 1–10.

[5] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to break an API: Cost negotiation and community values in three software ecosystems. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 109–120.

[6] Martin Brandtner, Emanuel Giger, and Harald Gall. 2014. Supporting continuous integration by mashing-up software quality information. In *Proc. Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*. IEEE, 184–193.

[7] Martin Brandtner, Sebastian C Müller, Philipp Leitner, and Harald C Gall. 2015. Sqa-profiles: Rule-based activity profiles for continuous integration environments. In *Proc. International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 301–310.

[8] Marco Caliendo and Sabine Kopeinig. 2008. Some practical guidance for the implementation of propensity score matching. *Journal of economic surveys* 22, 1 (2008), 31–72.

[9] José Campos, Andrea Arcuri, Gordon Fraser, and Rui Abreu. 2014. Continuous test generation: enhancing continuous integration with automated test generation. In *Proc. International Conference on Automated Software Engineering (ASE)*. ACM, 55–66.

[10] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 186–196.

[11] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. 2014. Challenges when adopting continuous integration: A case study. In *Proc. International Conference on Product-Focused Software Process Improvement*. Springer, 17–32.

[12] 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.

[13] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 235–245.

[14] Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. 2017. Trade-offs in continuous integration: assurance, security, and flexibility. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 197–207.

[15] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *Proc. International Conference on Automated Software Engineering (ASE)*. IEEE, 426–437.

[16] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. Vol. 398. John Wiley & Sons.

[17] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, 92–101.

[18] Eric Knauss, Miroslaw Staron, Wilhelm Meding, Ola Söder, Agneta Nilsson, and Magnus Castell. 2015. Supporting continuous integration by code-churn based test selection. In *Proc. Proceedings of the Second International Workshop on Rapid Continuous Software Engineering*. IEEE Press, 19–25.

[19] Eero Laukkanen, Maria Paasivaara, and Teemu Arvonen. 2015. Stakeholder Perceptions of the Adoption of Continuous Integration–A Case Study. In *Proc. Agile Conference (AGILE), 2015*. IEEE, 11–20.

[20] Paul R Lawrence and Jay W Lorsch. 1969. Developing organizations: Diagnosis and action. (1969).

[21] 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* 32, 2 (2015), 64–72.

[22] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. 2013. An empirical study of api stability and adoption in the android ecosystem. In *Proc. International Conference on Software Maintenance (ICSM)*. IEEE, 70–79.

[23] Daniel McFadden. 1977. Quantitative Methods Tor Analyzing Travel Behaviour Of Individuals: Some Recent Developments. (1977).

[24] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. 2012. Climbing the" Stairway to Heaven"–A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *Proc. Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 392–399.

[25] Jagdish K Patel, CH Kapadia, Donald Bruce Owen, and JK Patel. 1976. *Handbook of statistical distributions.* Technical Report. M. Dekker New York.

[26] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. 2016. Continuous deployment at Facebook and OANDA. In *Proc. International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 21–30.

[27] Gerald Schermann, Jürgen Cito, Philipp Leitner, and Harald C Gall. 2016. Towards quality gates in continuous delivery and deployment. In *Proc. International Conference on Program Comprehension (ICPC)*. IEEE, 1–4.

[28] Jens Smeds, Kristian Nybom, and Ivan Porres. 2015. DevOps: a definition and perceived adoption impediments. In *Proc. International Conference on Agile Software Development*. Springer, 166–177.

[29] 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 *Proc. International Conference on Software Engineering (ICSE) (ICSE)*. ACM.

[30] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. 2016. The sky is not the limit: multitasking across GitHub projects. In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, 994–1005.

[31] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 805–816.

[32] 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 *Proc. International Conference on Automated Software Engineering (ASE)*. 60–71.