

Team Discussions and Dynamics During DevOps Tool Adoptions in OSS Projects

Likang Yin

DECAL and CS Department
University of California, Davis
lkyin@ucdavis.edu

Vladimir Filkov

DECAL and CS Department
University of California, Davis
vfilkov@ucdavis.edu

ABSTRACT

In Open Source Software (OSS) projects, pre-built tools dominate DevOps-oriented pipelines. In practice, a multitude of configuration management, cloud-based continuous integration, and automated deployment tools exist, and often more than one for each task. Tools are adopted (and given up) by OSS projects regularly. Prior work has shown that some tool adoptions are preceded by discussions, and that tool adoptions can result in benefits to the project. But important questions remain: how do teams decide to adopt a tool? What is discussed before the adoption and for how long? And, what team characteristics are determinant of the adoption?

In this paper, we employ a large-scale empirical study in order to characterize the team discussions and to discern the team-level determinants of tool adoption into OSS projects' development pipelines. Guided by theories of team and individual motivations and dynamics, we perform exploratory data analyses, do deep-dive case studies, and develop regression models to learn the determinants of adoption and discussion length, and the direction of their effect on the adoption. From data of commit and comment traces of large-scale GitHub projects, our models find that prior exposure to a tool and member involvement are positively associated with the tool adoption, while longer discussions and the number of newer team members associate negatively. These results can provide guidance beyond the technical appropriateness for the timeliness of tool adoptions in diverse programmer teams.

Our data and code is available at https://github.com/lkyin/tool_adoptions.

ACM Reference Format:

Likang Yin and Vladimir Filkov. 2020. Team Discussions and Dynamics During DevOps Tool Adoptions in OSS Projects. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20)*, September 21–25, 2020, Virtual Event, Australia. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3324884.3416640>

1 INTRODUCTION

OSS software development practices are evolving away from *de novo* programming and toward adopting pre-made tools for various

tasks, which are then integrated into existing development and production pipelines [14, 50]. Part of the reason for this has been the popularity of the DevOps software development movement, which seeks to bring changes into production as quickly as possible without compromising software quality, primarily by automating the processes of building, testing, and deploying software. In practice, DevOps is supported by a multitude of configuration management, cloud-based continuous integration, and automated deployment tools, short-circuiting the need for coding from scratch [22, 24]. Using pre-made tools can shorten the development process, so long as an appropriate set of tools is used and properly integrated into a development pipeline.

Tool adoption decisions, however, are often not well informed. DevOps engineers frequently lack the decision-making support to help them discern the best choices among the many tools available [24]. In large part, that's because current empirical evidence on the effectiveness of DevOps practices is, at best, fragmented and incomplete. While questions about best tools and practices in DevOps abound in online forums, the existing answers are typically generic rules of thumb, or dated advice, mostly based on third-party experiences, often non-applicable to the specific context. While they likely consider that scattered knowledge, teams seem to leverage their strengths and experiences in making tool adoption decisions. Some tool adoption events in projects are preceded by a discussion among team members on the issues involved [24], but it is not clear what is discussed in them among team members and how those discussions correlate with adoption decisions.

Moreover, instituting a project-wide change is a complex social process when there are many stakeholders [44]. Adopting a new tool, e.g., can require a team-wide adjustment in practices that affect every developer—thus they are all stakeholders in the adoption decision. This is especially true when the team is more diverse in terms of developer tenure with the project, their prior exposure to the tool being considered for adoption, and their day-to-day involvement in the project. Naturally, supporters and detractors can and do arise over decisions when developers espouse different views toward a tool, resulting in champions and detractors, and sometimes arguments can get emotional [20, 27]. Many of these individual and team-level factors may contribute to the ultimate adoption decision, but which ones actually do? And in what proportion?

Inspired by the availability of large, comprehensive data sets on tool adoptions from diverse GitHub projects, here we undertake both qualitative and quantitative methods to uncover the social determinants of team discussions and dynamics leading to tool adoption events in OSS projects. We operationalize our study at the team-level instead of at the individual level. Central to our study is the analogy that a project team adopting a tool is akin to

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 ACM 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.

ASE '20, September 21–25, 2020, Virtual Event, Australia

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-6768-4/20/09...\$15.00

<https://doi.org/10.1145/3324884.3416640>

a community adopting an idea. We use theories on diffusion of innovation and individual and team social judgment to guide us in discerning the important factors underlying tool adoption.

We start from a data set of social and technical traces from a large number of GitHub projects, together with project-tool adoption data for each. Then, we perform exploratory data analyses, do deep-dive case studies, and built regression models to determine how team properties and their communication behaviors are associated with tool adoptions. We find that:

- Team dynamics changes around adoption time in substantial ways, and some of those changes remain with the project.
- Project teams undergo meaningful discussions and exhibit significant dynamics changes in the period before and after a new tool is adopted.
- Influencer developer's participation is associated with shorter discussion length, and likelier tool adoption.
- New developers are positively associated with longer discussion length and lower adoption success.

Related questions have been asked before, in narrower settings. Zhu et al. [54] use code accept/ignore rate to compare the goodness between issue tracker and pull request systems, but less focus on the adoption dynamics. Using surveys Xiao et al. [52] find that coworker recommendation is a significant determinant of security tool use. Witschey et al. [51] find that the strongest predictor of using security tools is the ability to observe their coworkers using those tools. These findings are based on a small number of commercial software projects and can be difficult to generalize, especially to OSS projects. Moreover, surveys, by their nature are based on individual experience and feedback. Kavalier et al. [24] have looked more comprehensively at a large swath of JavaScript projects, but have focused more on the effects of tool choices on software engineering outcomes, and found that some choices are better than others.

2 BACKGROUND AND THEORIES

Here we position our current work in the space of representative prior work on tools, tool adoptions, and OSS teams, as well as present two most relevant sociological theories that will guide our hypotheses and research questions.

2.1 Tools, Teams and Adoptions

Software developers use various techniques to implement and maintain software, including, at times, adopting new technology [7]. T. Gorschek et al. [12] describe five different stages in technology transfer, including identifying potential problems, formulating issues, proposing candidate solutions, validations, and releasing solutions. S.L. Pfleeger [32] proposed a model of technology transfer that can be tailored to a particular organization's needs. Riemen-schneider et al. [35] find that opinions of developers' coworkers and supervisors on using some technology matters to an individual developer when they consider whether to use this method. Our paper extends such implications to tool adoptions in OSS projects.

Marlow et al. [29] find that length of tenure may be associated with attitudes toward new tools. More senior developers may get more attached to a working style, thus making them less flexible to adopt new tools. New developers [10], in contrast, need less

time to adapt to a new working style due to lack of history in the project, and eagerness to learn/follow technology trends. Xiao et al. [52] find developers are more likely to adopt a tool if their peers or co-workers are using or have used it. Also, if experienced individuals join a project, their knowledge and social ties move with them to the project [1, 49]. Thus, the sum-knowledge of the whole team is constantly changing as the project evolves. Likewise, previous research has found that emotional contagion from co-workers can percolate within the team [15, 39] causing a slowdown in communication and productivity.

With publicly available traces of working records and discussions, it is possible for researchers to study team dynamics before, during, and after a change. However, studying tool adoptions is complex because of the variety of contributing factors. Previous work [16] has found that changing previous practices can force software developers out of their safe zone, resulting in nonconformity with the methodology. Zelkowitz et al. [53] find that many software professionals are resistant to change, and that timing is of importance when adopting new technology. Johnson et al. [21] consider that having intuitive defect presentation may contribute to the willingness of using static analysis tools. Related literature [51, 52] shows that developers' social networks (e.g., through discussions about tools [43]) benefit the spread of tools. Poller et al. [33] pointed out correlations between organizational factors and the success of security practices.

2.2 Diffusion of Innovations Theory

Diffusion of innovations (DOI) theory was first proposed by Rogers in 1983 [36], and has since become popular in socio-technical fields. Rogers considered diffusion as the process by which an innovation unfolds over time through communication channels among the population, and found that some properties of innovation are crucial to an adoption: perceived usefulness, perceived complexity, peer pressure, etc. The innovation diffusion process at an organizational level can be summarized as having two stages [36, 40]: initiation (perceiving the issues, and knowing they can be addressed by adopting certain innovations), and implementation (adopting the innovation and customizing it to fit own culture if necessary, then the team-collected information reinforces/devitalize the adoption until the innovation becomes a part of the organization).

In the context of software engineering, DOI theory can be of vital value to offer understandings of the phenomenology and consequences. E.g., tool builders want to know if anyone will use the tool they built, and how to persuade the community to use it. For general developers, they want to know if there exist tools to help them be more efficient. However, even if a tool can be beneficial, individual resistance may still exist. To reduce such individual resistance, organizational mandate has been shown to have influence on adoptions [16, 35]. However, those authors also find that the organizational mandate is not sustainable compared to other factors. A catalog of non-coercive adoption patterns has been proposed [40], to help organizations achieve successful software engineering practices in a more persuasive manner.

In GitHub OSS projects, Bertram et al. [2] found that communication and knowledge sharing exist for coordinating work in issue

trackers. This suggests that developers within a project communicate and learn from each other [45]. Moreover, knowledge sharing exists even across projects. Singer et al. [40] noted that some developers use GitHub to learn how other projects use the same tools in their projects. Thus, developers in the GitHub ecosystem, and in its tighter sub-ecosystems, share and distribute knowledge about tools through participation in different projects, thus creating a diffusion process.

2.3 Social Judgment Theory

Social Judgment Theory (SJT) [4] was proposed to study how people self-persuade to adopt a new idea when they encounter it. According to SJT, when people are exposed to new information or a new environment, they tend to consider three things. First is their previously formed attitude, or *anchor* to which they compare the new idea [46]. Then they look at available *alternatives*. In this process, people recognize themselves, form their views, and express their ideas. Finally, there is the *ego involvement* or the centrality of the issue being considered to a person's life, which can explain why people can accept some ideas and novelty easier than others. Individuals with high ego involvement on an issue tend to be more passionate on the topic and are more likely to evaluate all possible positions [13]. High-ego individuals also have a larger latitude of rejection, and it is difficult to persuade them to adopt a new idea. In contrast, low-ego individuals tend to have a larger non-commitment latitude, meaning they often do not take a stance on an issue, and they do not care much about the arguments.

There are several ways to aggregate the judgments of individuals from a group into a *group judgment* [11, 37]. *Mathematical aggregation* amounts to simple counting/averaging of individual judgment. On the other hand, *behavioral aggregation* is the outcome of group members agreeing after discussing the matter. Experimental evidence suggests that group judgment is generally more accurate than individual judgment, and how it is measured can be significant to the outcome [11]. However, others found that the superiority of group judgment is due to reliability produced by larger samples [25].

In the context of OSS projects, team discussions on tool adoptions transpire during which possible options are proposed. These discussions can be considered a form of behavioral aggregation of individual opinions. However, group judgment may get biased, as high-ego individuals with strong opinions can potentially sway a group decision in their direction, even if it offers no overall benefit.

3 HYPOTHESES AND RESEARCH QUESTIONS

Central to this paper is the analogy that adopting a tool is akin to adopting an idea. An OSS project involves a social organization in which activities are coordinated through communication to achieve both individual and collective goals. By coordinating activities, the organizational structure has to be created to assist individuals to communicate. In GitHub projects, in particular, the two main communication channels are through code committing and issue posting. Moreover, contributing code changes and approving others' pull requests suggests that the developers are mutually aware of each other, and this forms the basis for communications and discussions.

The DOI theory suggests that innovations and new technologies can be spread to teams through diffusion. For tool adoption in GitHub projects, this diffusion happens through information exchange within and between projects, through their developers. One mechanism is through reading/participating in discussions that are accessible to all team members in a project. The publicly available traces of commits and comments allow us to study discussion dynamics of how tools are perceived, discussed, and then finally adopted. Thus, a hypothesis arising from DOI is that developers who have previously had exposure to certain tools will be more knowledgeable of them and contribute to a discussion on it, to make the adoption process smoother and faster.

On the other hand, SJT suggests that since it is likely some developers prefer adopting a tool more strongly than other developers do, the former naturally will have high-ego on adoptions. A hypothesis arising is that developers who post more comments on tools than others will be the influencers in the discussions, and will affect the direction of the adoption. Moreover, contributing to cognitive dissonance, people will react more strongly to negative information than positive information [42]. Therefore, another hypothesis is that the people's discussion sentiment (positive or negative) may correlate with eventual tool adoption.

We formalize the above into our Research Questions (RQs), as follows. First, we seek to uncover the patterns and changes happening at team-level, in the period surrounding tool adoption events.

RQ1: What is the team dynamics when a project team goes through the process of adopting a new tool?

Then, we want to understand what goes on in the discussions before and after the tools are adopted.

RQ2: What topics are discussed during tool adoptions? How are people's sentiments evolving toward the tools they are adopting?

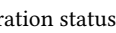
Next we look at notable individuals in the discussions. According to the aforementioned theories, people who care the most and comment voluminously, i.e. influencers, may play a significant role in the innovation diffusion process. Namely,

RQ3: Are tool adoption events associated with influencers? How much does their opinion weigh in on others?

In the final thrust, to comprehensively understand tool adoption and discussions, we quantitatively model adoptions and discussion length in terms of our chosen variables using multiple regression.

RQ4: What are the quantitative determinants of project-wide tool adoption and the preceding discussion length?

4 DATA

We start from a data set by Trockman et al.[48] of GitHub *npm* projects that have adopted any of 19 different DevOps tools¹. They were collected by gathering tool badges from the projects' README.md files. Some projects use badges to signal important information [34], e.g. code coverage percentage , number of downloads per month , package dependencies , and continuous integration status . The tool adoption data is current as of Jan 2019. The data set consists of 52,923 distinct GitHub projects, the adopted tools, and the adoption dates. In total, 96,176 tool adoptions are identified, or about 2 tool adoptions per project on average. Among them, 28,430 projects

¹Data and code is available at https://github.com/lkyin/tool_adoptions

Table 1: Tool adoption event summary

Tool	Task Class	Per Tool	Per Tool Category
karma	Browser	4116	6157
sauce		1654	
selenium		387	
coveralls	Coverage	14430	21626
codecov		4239	
codeclimate		2731	
codacy		213	
coverity		13	
uglify	Minifier	2018	2124
minimist		62	
minifier		44	
mocha	Testing	33280	47119
istanbul		11864	
jasmine		1975	
eslint	Linter	10978	18969
standard		4827	
jshint		3164	
snyk	Dependence Manager	179	181
gemnasium		2	

adopted only one tool, 11,272 projects adopted two tools, 8,900 projects adopted three tools, and 3,378 projects adopted four tools. Projects which have adopted more than four tools are fewer than 2% of the total.

The collected tools can be classified into the following six categories according to their functionality. We illustrate each category with an example tool. **Browser testing**: e.g., *Selenium* is an automated testing framework for automated testing of web applications and User Interfaces (UIs) [6]. **Test Coverage**: e.g., *Coveralls* measures software quality in terms of test cases line coverage, function coverage, branch coverage, and statement coverage [17]. **Minifier**: e.g., *Uglifyjs* is a JavaScript compressor tool used to merge and minimize JS resources by removing blank rows, shorten the variables and functions names to make the web applications load faster [41]. **Testing**: e.g., *Mocha* has good support for testing asynchronous code, allowing any use of failed exception test libraries [9]. **Linters**: e.g., *ESLint* is a plug-in JavaScript code style/error detection tool [47], thereby achieving effective control of the quality of the project. **Dependency managers**: e.g., *Snyk* helps developers track the dependency tree to find which module introduces the vulnerability [30, 38]. The categories and the summary statistics of adoption data is given in Table 1.

4.1 Data Collection and Cleaning

We use the *GitHub API (v3)* [8] to collect and extract historical records of the commits and discussions from the GitHub projects. For commits data, the author is the one who made the changes, while the committer is the one who committed the changes to GitHub. Thus, we align the ‘author’ with each commit. For discussions, since the commit messages are usually not meaningful [19], we only collect and use issues and comments as their discussions.

Some projects have very low levels of activity, while others have team sizes that may be too small to study their dynamics. To avoid those, in this paper commits and comments of only the durable and persistent projects are collected. We set three requirements: (1) *Durable*: The projects that have commit records spanning at least

two years. (2) *Persistent*: The projects having at least 6 months of commit history before and after the tool adoption event and have at least 50 total commits. (3) *Related*: The projects having at least one comment related to the adopted tools. After filtering, the final data set consists of 684 distinct GitHub projects.

Since we focus on discussion comments which specifically associate with tools, we filter out all discussion comments which do not explicitly mention a tool name (of the 19 tool names), with the following exception. If an issue starts a thread and mentions a tool name, the replies to that thread are likely a part of the discussion, therefore, all comments following the thread are also included even if they do not contain a tool name. On the other hand, some issues are automatically posted, e.g., by continuous integration (CI) tools; we identify the comments by checking the title of the comments (e.g., ‘[Snyk Update]’) and exclude the comments from the data set.

We identify and remove the following strings from comments that do not comprise text, as they would bias downstream analysis: non-ASCII characters (by checking if all characters are from ‘u4e00’ to ‘u9fff’), code snippets (by checking if the comments are enclosed with single or triple backticks), URLs (by using the regular expression from the *re* module in *Python 3.7* with pattern ‘https?://S+’), and emojis by checking the encoding of the characters. Finally, since some comments mention multiple tools at the same time, therefore, they are counted multiple times. To avoid such duplication by those comments (about 1.77% of the total), we manually re-label these comments with only one tool, we achieve this mainly by referring the outcome of the adoption (which one is finally adopted) and context of the discussion.

4.2 Variables Used

The variables that we use in this study have been identified based on our discussion and consideration of the underlying theories. They include the following.

Outcomes: Adoption success and discussion length. *Adoption success* (*adoption_success*) is a binary variable (0="No" or 1="Yes") indicating whether a tool was adopted in a project. A successful adoption is if a tool is being used in a project, regardless of whether discussions on it ever happened. An unsuccessful adoption is if a project’s team had a discussion on a tool yet they never adopted it. *Discussion length* (*discussion_length*) for a project and a tool is calculated as the number of months from the first day the tool was mentioned in the project discussion, until the tool was adopted. The discussion length suggests how long it takes for teams to reach an agreement and eventually adopt the tool, although in practice can be much longer if the team keeps coming back to discussing a tool after longer breaks without mentioning it. To address those cases we introduce a control variable *num_mentions*, see below, which measures the volume of tool mentions in a discussion.

Controls: Project Age, Number of Commits, Number of Comments/Mentions. *Project age* (*project_age*) at the time of adoption of a tool is the number of months from the first commit date in the project to that specific tool’s adoption date. *Number of Commits* (*num_commits*) at the time of adoption is the number of commits made during the discussion (based on the discussion length above) on that tool, in the project. *Number of Comments* (*num_comments*) / *Mentions* (*num_mentions*) is defined as the number of comments

Table 2: Summary statistics for our 10 variables over all 1,085 adoption events (after removal of top 2% as outliers)

Statistic	Mean	St. Dev.	Min	Max
adoption_success	0.798	0.402	0	1
discussion_length	9.927	11.428	0.033	56.633
project_age	33.542	20.688	6	100
num_comments	15.852	32.792	1	272
num_commits	338.246	588.521	0	4,163
num_new_dev	18.852	32.933	0	252
num_w_tool_expos	7.724	13.973	0	133
num_involved_dev	2.373	2.377	1	23
num_neg_dev	0.678	1.077	0	7
num_pos_dev	0.641	0.975	0	8

(including all follow-up comments in the same thread) made in that tool’s discussion interval, while the number of mentions only counts the number of comments which explicitly mention the tools, for a given project. *Tool* (*tool*) is the full name in lowercase of the corresponding tool (e.g., *eslint*).

Team Metrics: We calculate team measures for each project, for each tool discussion. *New Developers* (*num_new_dev*) at time *t* in the project are those developers who have made their first contribution (either by committing code changes or participating in discussions) within the 3 months prior to *t*. For convenience, we refer to all other developers who are not new developers as *Senior developers*. *Developers with prior tool exposure* (*num_w_tool_expos*) are the developers who had already been in a project before time *t* that had used that tool, and have committed code changes before time *t* to current project (but after their contributions to the other, tool using project). In contrast, the developers without prior exposure are the ones who did not participate in a project that had used the tool before time *t*. *Involved Developers* (*num_involved_dev*) are the developers who have been involved (i.e., participated) in the tool discussion. *Positive Developers* (*num_pos_dev*) are the developers who have posted overall more comments with positive sentiment than negative sentiment in the tool discussion. While *Negative Developers* (*num_neg_dev*) are the opposite. The descriptive statistics for the metrics are shown in Table 2.

5 METHODS

5.1 Topics Identification

We use Latent Dirichlet Allocation (LDA) [3] to study topics in discussions. LDA is a statistical technique used to identify topics in large documents and high-frequency words associated with the topics. LDA yields a topic probability distribution for each document, enabling topic clustering and/or text classification across all documents in a set.

Before training the LDA model, we pre-processed the GitHub comments by tokenizing with the Apache Open NLP library [26], and stemming with the Porter stemmer (this removed all stop-words from the comments). Due to the large number of discussions we have, many high-frequency words are not very meaningful. To get a corpus with a higher concentration of topics [28], we removed

both the 5% words with the lowest frequency (e.g., user names) and the 5% words with the highest frequency (e.g., bugs). After this, the LDA model is more able to distinguish topics from each other.

5.2 Sentiment Analysis

A sentiment analysis tool identifies the emotional characteristic of a text, typically in terms of its aggregate positivity or negativity. Many sentiment analysis tools used in software engineering are trained solely on social media corpora, e.g., Twitter, Facebook, and Yelp. However, some words in the context of software engineering can represent a different meaning compared to social media corpus [23, 39]. For example, to ‘kill a process’ is neutral in the context of programming, while some sentiment analysis predictors trained on other media corpus, treat it as a strong indicator for negative emotion. To avoid such issue, we use Senti4SD to predict the sentiment of the GitHub comments. It has been trained on Stack Overflow annotated comments, and it has been shown to be more accurate in the software engineering domain [5, 18]. Senti4SD yields ternary sentiment for each comment, i.e., positive, neutral, or negative. The following are examples from our data of positive, neutral, and negative comments, as per Senti4SD (sensitive words are anonymized and replaced by <notation>). *Positive*: “sure <user>! appreciate your point, thanks for the suggestion.” *Neutral*: “Let’s have this project actually be <tool> and not try to duplicate on our own what <tool> does internally. Let’s just let <tool> handle all merging itself.” *Negative*: “BTW, <tool> failed because you added a function without tests.”

GitHub discussion comments can still be different from the Stack Overflow comments. To verify that Senti4SD can effectively identify sentiment in comments on GitHub, we selected a random set of 50 comments and via observation determined them to contain 25 comments of neutral, 13 comments of negative, and 12 comments of positive sentiment. Then we ran them through Senti4SD and found that 6 were mispredicted by Senti4SD, showing the accuracy of 88% (2 neutral comments were deemed negative, while 1 positive and 3 negative comments were deemed neutral).

We aligned each post with their Senti4SD derived ternary sentiment (i.e., positive, neutral, and negative). We find that 23.8% of the discussions are positive, 48.4% of the comments are neutral, and 27.8% of them are negative. The average length of the comments is 194 characters for neutral, 242 characters for positive, and 422 characters for the negative. This suggests that the negative comments may carry more information than other two types.

5.3 Linear Mixed-effect Regression

We use Generalized Linear Mixed Effect Regression (GLMER) models (*glmer* package in *R*) to study the contribution of our independent variables to explain the variability in the outcome variable, while mixing fixed and random effects. The *tool* is used as the random effect in the models.

We use logistic regression for modeling the adoption success and generalized regression with the Poisson family for the modeling discussion length. To avoid convergence issue we use the *bobyqa* optimizer. We use *scale()* function to z-normalize each variable as they vary across orders of magnitude between variables. To avoid influential points caused by outliers in the fixed effects, we remove

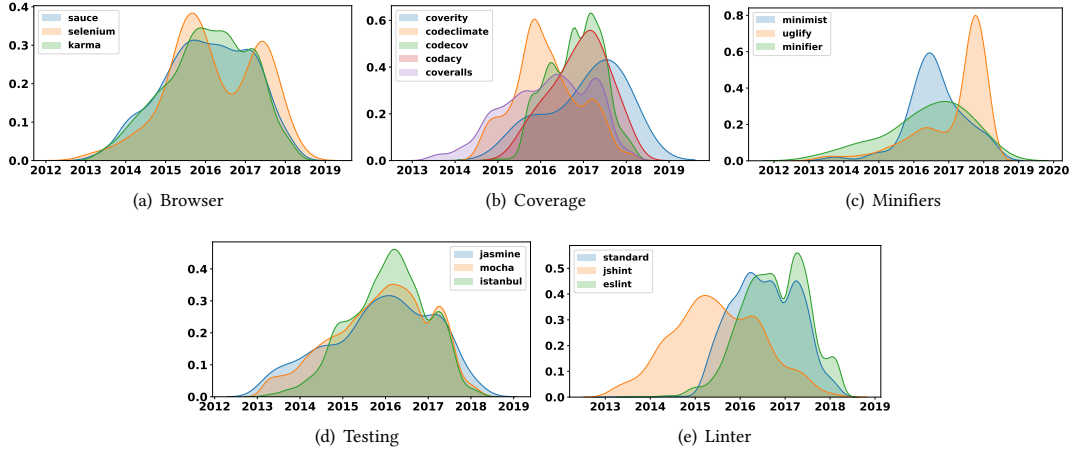


Figure 1: The adoption time distributions of tool adoption events of five tool categories.

the projects with top 2% *num_commits*, *num_comments*, and with top 1% of the rest variables. We also use nonlinear *log* transform for two variables: *num_new_dev* and *num_involved_dev*, containing extreme values and high variance. We use the Variance Inflation Factor (VIF) to check whether multicollinearity exists between the independent variables in the regression models, which can lead to regression coefficients that are difficult to interpret. Typically, if the VIF values are smaller than 5 then multicollinearity is not significant. This is the case with all our models. To describe the goodness of fit of our models, we use the *squared GLMM()* function in *R* to report two pseudo- R^2 values: the marginal R^2 , interpreted as the variance solely described by the fixed effects, and the conditional R^2 , interpreted as the variance introduced by both fixed and random effects in the model [31]. We also report the standard goodness of fit measures of log-likelihood and the Bayesian information criterion, the latter often used for model choice.

6 RESULTS AND DISCUSSION

6.1 RQ1: Tool Adoption and Team Dynamics

First, to study tool adoption events over time, we align all projects around their adoption dates and plot those adoptions for each tool. The results, per tool category and per tool are shown in Figure 1. The figures suggest that adoptions in OSS projects spread in a non-constant speed, some group of people adopt a tool much sooner than the average adoption time, while others adopt it only if they are fully convinced, i.e., have an adoption lag. This fits well within the predictions of the DOI theory. Note that the tool category *Dependence manager* is not included because of lacking enough data points.

Second, to study the team dynamics around adoption events, we examine the temporal data of our five team-level metrics: the number of developers with prior exposure, new developers, involved developers, comments associated with tools, and commits at monthly intervals, as illustrated in Figure 2.

We see from Figure 2 that the average number of involved developers is almost linearly increasing over time, likely correlating

Table 3: Topics Discovered in Discussions

Topic	Sample vocabulary
1 Testing	run, test, use, report, case, mocha
2 Development	js, setup, window, browser, npm
3 Debugging	fail, stack, error, timeout, check
4 General ideas	work, support, dependency, need
5 Integration	CI, function, module, nodejs, client

with the general GitHub trend. This implies that more developers participate in the discussions of adopting tools.

We also observe a significant discontinuity in the steady numbers of commits and comments just before and even more so after the adoption event, in the positive direction. This is arguably associated with increased activities related to the adoption. Also notable, is that the number of developers with prior exposure to the tool is steadily growing in the period before the adoption, thereby likely increasing the diffusion probabilities and the adoption chances.

Answer to RQ1: Successful adoption distributions are in line with DOI theory, with some projects adopting early and others late. We observe that new developer numbers increase slower than involved developer numbers but significantly more so after the tool adoption.

6.2 RQ2: Discussion Topics and Sentiment

Pre- and Post-adoption Discussions LDA is regularly used to reveal the frequent topics in text information. However, different from the corpus gathered from social media, the comments from GitHub, by their nature, are in a much narrow domain. Therefore, the number of different discussion topics on GitHub is much fewer compared to social media corpus. Even though, many topics still overlap with each other. We uncover the top-ranked topic features and their associated sample vocabulary in the tool discussions, and summarized them in Table 3.

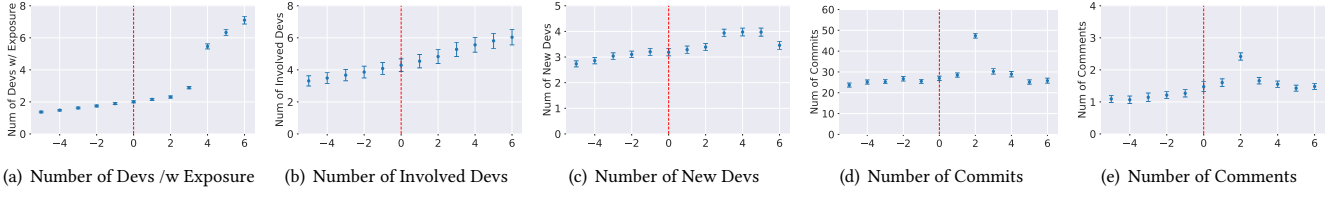


Figure 2: The monthly aggregated numbers of five variables (x-axis unit is in month), relative to the adoption month ($x = 0$), over all projects. Error lines show the ± 1 standard error away from the mean.

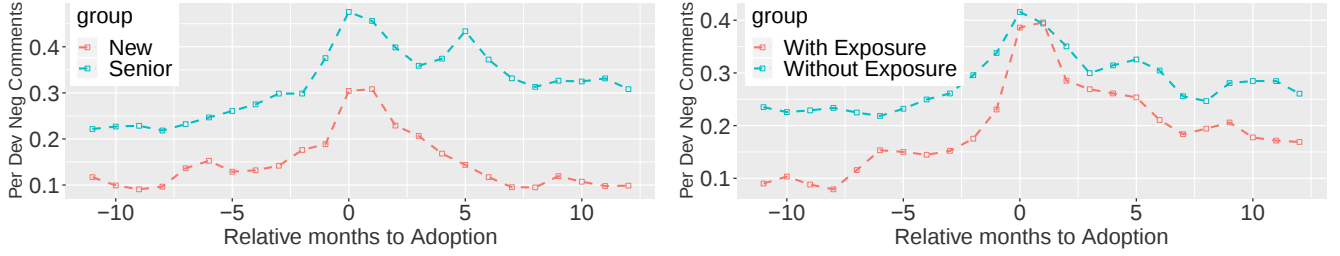


Figure 3: The per-developer negative comments posted by new developers and senior developers (left), and developers with prior tool exposure and developers without prior tool exposure (right). Adoption event happens at month $x = 0$.

To understand and identify topics and emerging patterns in the discussions, we go through 3,342 discussion comments within the 6 months prior to the adoption date. By using the topic-vocabulary pairs as the keywords, we find the following three topics in tool discussions are prominent:

Perceiving Demands of Tools: "right now the <file> is extremely distracting and the test output is impossible to read (just the result). We need a <tool> PR to solve this."

Choosing One Tool Over Another: "<dev 1> thanks. I should have done this, to begin with. I set up <tool 1> because that's what <tool 2> was using before. But you are correct, that is better for testing."

Deciding When to Adopt: "Thank you for sending us these contributions! Moving to <tool> is, in fact, something we have hoped to do, I just wanted to let you know it might take us a few more days before we're ready to engage."

We also analyze 4,278 discussions from the same projects set for the 6-month post-adoption period. The two identified prominent topics of discussions are about:

Adoption Feedback: "Not sure how I to write a test for this. I can't figure out how to get the output from <tool>"

Switching Tools: "But when comparing/choosing a testing framework you will definitely need to say one is better for you or a project at some point. I would be open to using <tool>."

In summary, we find the following prominent patterns exist in the tool-related discussions: a discussion seems to be initiated by an individual who found an issue and asked for addressing such issue with a tool, and it ends with an individual who has previously used similar tools presenting their experience, by them recommending a tool to adopt. We also find that discussion threads are goal-oriented, well structured, and proceed logically, and they are likely to be beneficial for developers to decide on which tool to adopt.

Developer Sentiment To test our hypotheses from SJT about ego involvement and from DOI about tool adoption, here we compare the sentiment in tool adoption discussions between the comments of a) new developers and senior (i.e., not new) developers, and b) between developers with prior tool exposure and the ones without prior tool exposure. To do that, we compare the negative comments posted separately by both new developers and senior developers. As shown in Figure 3(a), before the adoption, new developers are much less negative to adopting new tools than senior developers, even though their negativity significantly grows just before and more so after the adoption events, if only for a short time. This is consistent with SJT: new developers have less involvement in the projects and thus lower emotional attachment than senior developers, while the latter have to (perhaps begrudgingly) adapt to the changes in the project. Even more interestingly, the negative sentiment of the senior developers persists, which can in the longer term affect project cohesion and effective management.

On the other hand, as shown in Figure 3(b), the per-developer negativity of developers with prior exposure is much lower than the developers without exposure, for all time-bins. And after implementing tool adoptions, the negativity of developers with prior exposure is very close to developers without exposure, however, the negativity of developers with prior exposure drops faster right after the adoption and remains lower in the long term. This validates the assumption that developers with prior exposure are less negative toward the tools than the ones without exposure. Moreover, we find that the curves of the two types of developers are similar to some degree, suggesting both are reacting to the same events, or are communicating together.

Relative sentiment. To understand why developers choose one tool t_1 over another tool t_2 from the same category (e.g., both



Figure 4: Negativity toward tools in the same category, illustrated both before the adoption (red) and after adoption (green).

t_1 and t_2 are linters), we compare the relative negativity of their corresponding discussions. First, we calculate the baseline negativity for each project (i.e., the ratio of negative comments over all comments). Then we calculate the aggregated negativity of a tool as the ratio of negative over all comments in the discussion, minus the project's baseline negativity. In Figure 4, each row represents a pair of two tools (t_1 - t_2) from the same category. The bars show the aggregated (team-level) negativity to tool t_2 of the projects before (red) and after (green) adopting tool t_1 . If tool t_1 and t_2 are the same (i.e., $t_1 = t_2$), the bar simply represents the change of negativity after adopting the tool t_1 (i.e., t_2). As shown in Figure 4(a), the bar *Jasmine-Jasmine* shows that, the post-negativity becomes more negative than pre-negativity, suggesting that developers find *Jasmine* more difficult to use than expected. In Figure 4(b), *standard-jshint* bar shows that after adopting tool *standard*, the sentiment toward *jshint* becomes less negative (i.e., more positive).

Answer to RQ2: We identify three most significant scenarios in the tool discussions: perception of tools, choosing a tool over another, and deciding when to adopt a tool. We also find that sentiment toward a tool is associated with developer seniority and prior exposure to the tool. Finally, sentiment toward a tool is associated with adoptions, and it can change after adoptions.

6.3 RQ3: Influencers and Adoptions

In the sentiment data set, we find that 5% of developers posted 35% of total negative comments, indicating there might exist some strong ego-involvement and possibly influencers during the discussion period. Here, we seek to answer if influencers exist, identify them, and see how they affect tool adoptions. We first define *influencers* as those developers who post more than 50% of the total comments in the tool adoption discussions. To have reliable data to study developers who frequently comment in each discussion, we decided to only consider adoption discussions with more than ten comments. That filter leaves us 592 distinct adoption events to study. Among them, 379 adoptions are successful, and 213 adoptions are unsuccessful. We define the Successful Adoption Rate (SAR) as the ratio of the successful adoptions to the total, i.e., $379/592 = 64.02\%$.

A very common situation is that a project member wants to have a testing tool for their project, and the developers create an issue to query other's opinions on whether to adopt this tool. If the majority of main contributors agree on adopting a testing tool, then they would be discussing which tool to adopt. Our hypotheses,

arising from the ego-involvement considerations in SJT, are that: 1) a project with an influencer has a higher likelihood to have adopted a tool, and 2) the higher the negativity of a strong influencer, the less likely it is for the tool to have been adopted.

We test the first hypothesis by comparing successful adoption rates in projects with strong influencers to those without. We find that for the former, the SAR is 68.86% and for the latter 57.75%.

To test the second hypothesis, among the adoptions that have had an influencer, we found that if the sentiment of the strong influencer was positive on the tool, the SAR is 72.18%, while if the sentiment of the strong influencer was negative, the SAR is 64.71%.

In contrast, and as predicted by SJT, without an influencer integrating the democratic opinions into a consensus decision may be more difficult and take longer, while the influencer can speed up the adoption process. We find that without an influencer, the average discussion length is about 15 months, while with an influencer, the length decreases to 13 months, on average.

Answer to RQ3: We identify strong influencers in the projects, and show that projects having strong influencers have more successful adoptions and shorter adoption discussions. Moreover, the sentiment of the strong influencer correlates with the adoptions.

6.3.1 Case Study. We give examples from two comparably sized projects *testem/testem* and *bower/bower* to illustrate how a strong influencer can be of help in tool adoptions. The project sizes are similar to each other (number of commits: 2,305 v.s. 2,726; number of contributors: 157 v.s. 209).

With a strong influencer In the project *testem/testem*, the following discussion transpired, on using a tool to automate testing, where <Dev 2> is the strong influencer in the team.

<Dev 1>: "... I think it would be convenient to render the page as template and pass there..."

<Dev 2> "Why do you need this? Please give more details about your use case."

<Dev 1> "... I use testem not just to run unit tests to see standard test report page, but as a watching tool that automatically reloads my web application when sources changed ..."

<Dev 2> "I get most of what you are saying. Are you using <tool 1>? The hash issue I think I need to rethink how to handle that ..."

<Dev 1> "Now I switched to <tool 2> (as it really more robust and convenient), I used <tool 1> as well it doesn't actually matter. And I use dependency management tool to load scripts ..."

Without a strong influencer In contrast, the project *bower/bower* encountered an issue when trying to use a tool for automated testing. A developer asked for help from the community, however, no one presented strong opinions in favor of continued use of this tool. One member suggested to not use <tool> anymore and switch to another tool.

<Dev 1>: "Should we have a common way to declare the tests for any component? For example, I'd specify <tool> and file in <file>. I'd then run <command>, which would open the <tool> page in a browser. Thoughts?"

<Dev 2>: "what we could support is something similar to <package name> which is common scripts specified in the <file> ... what do <Dev 3> and <Dev 4> think?"

<Dev 4>: "<Dev 5> has said it was a mistake to make them all globals. they should be triggered with <file> ..."

<Dev 6>: "... Any component with unit tests that I've written just ends up using a separate node module (like <tool>) to automate the test workflow. I'd be in favor of closing this."

The two decisions are in contrast. In the first project, tool adoption happened after a strong influencer insists on it. In the second, multiple project members are involved, and the project adopted a different tool than the one discussed.

6.4 RQ4: Adoption & Discussion Determinants

In the previous RQs, we conducted exploratory and qualitative studies of team discussion and dynamics before and following an adoption event. Here we triangulate those with quantitative studies, to understand the determinants, as well as the direction of their effects, on adoption success and discussion length.

Our data is naturally hierarchically organized based on the tool being discussed. We use *tool* as a random effect in our models, allowing all projects adopting a specific tool to have the same random intercept. All other variables are used as fixed effects in our mixed effect models.

We model each of the two outcomes with a base model, comprised of the control variables, and a full model, which adds to the base the complement of team variables. We perform the likelihood ratio test between the base and full models using the *anova()* function, and present both models for each outcome variable.

Modeling Adoption Success. The results are shown in Table 4. We see from the AIC that the full model fits the data significantly better than the base model, with the three significant team variables explaining about 5% of the total variance, as per the marginal R^2 . Overall, the fixed effects alone do not present a good model, but together with the random effect, the model is much better, at 48% conditional R^2 . This, together with the vif's being smaller than 5, gives us confidence that we can interpret the coefficients of the variables.

Of the controls, the variables *discussion_length*, *num_comments* and *num_commits* have significant, sizeable negative effect on tool adoptions, holding all else constant. This is consistent with the SJT prediction that group judgment needs more time to form in larger teams, and that an adoption may be more difficult to succeed since an agreement is needed from more people. *num_mentions*, on the other hand, shows a sizeable positive effect, which makes sense from a DOI perspective, that an adoption needs a wider spread to succeed.

Table 4: adoption_success glmer model, tool as random effect.

	Base Model	Full Model
scale(discussion_length)	-0.477*** (0.110)	-0.484*** (0.129)
scale(project_age)	-0.051 (0.105)	-0.115 (0.108)
scale(num_mentions)	0.612*** (0.150)	0.448** (0.214)
scale(num_comments)	-0.183* (0.110)	-0.268** (0.115)
scale(num_commits)	-0.208** (0.097)	-0.222** (0.108)
scale(log(num_new_dev + 0.1))		-0.464*** (0.163)
scale(num_w_tool_expos)		0.617*** (0.141)
scale(log(num_involved_dev + 0.1))		0.419** (0.181)
scale(num_pos_dev)		-0.126 (0.141)
scale(num_neg_dev)		-0.120 (0.140)
Constant	1.264*** (0.410)	1.434*** (0.399)
Observations	1,085	1,085
Log Likelihood	-407.502	-391.491
Akaike Inf. Crit.	829.003	806.981
Bayesian Inf. Crit.	863.929	866.854
Marginal R^2	9.94%	14.79%
Conditional R^2	46.72%	47.97%

Note: *p<0.1; **p<0.05; ***p<0.01

Of the team variables, *num_involved_dev* is positively associated with adoption success, all else held constant. When multiple developers are highly involved in the project, they may all be on the same page concerning the project's needs. This is consistent with SJT, as aligned egos will easier agree. The predictor *num_w_tool_expos* also has a significant, sizable positive effects on adoption success. One possible reason for this is that the developers who had previously been active in projects that have used the tool, are more familiar with the tool. Consistent with DOI theory, those are the developers that contribute to the diffusion (spread) of information on the tool in their new projects, which can lead to successful adoptions. Refined temporal diffusion models can offer a more detailed, temporal view of this diffusion process, and are left for future work. An interesting finding is that *num_new_dev* is significant and negatively associated with adoptions. We can see several explanations. First, new developers may not feel comfortable to state their opinions publicly, which practically, may amount to a negative overall opinion. Second, as they do not understand the ins and outs of the projects yet, they may not perceive the need for the change, especially since they have just recently started contributing. Also, we find that neither *num_neg_dev* nor *num_pos_dev* have a significant effect on adoptions. We see this in the context of SJT: high ego developers are likely to be the ones participating in the discussions, and their arguments, emotional or not, are unlikely to change the opinions of other high ego developers.

Modeling Discussion Length. As we see in Table 5, the AIC tells us that the full model fits the data significantly better than the base model, with the three significant team variables explaining about 40% of the total variance, as per the marginal R^2 . Overall, the fixed effects alone present a good model, but together with the random effect, the model is excellent, at 91% conditional R^2 . This, together with the vif's being smaller than 5, gives us confidence that we can interpret the variables coefficients and trust the model.

Table 5: *discussion_length* glmer model, tool as random effect.

	Base Model	Full Model
scale(project_age)	0.360*** (0.010)	0.242*** (0.010)
scale(num_mentions)	0.038*** (0.009)	0.059*** (0.014)
scale(num_comments)	0.072*** (0.009)	-0.072*** (0.010)
scale(num_commits)	0.183*** (0.008)	-0.006 (0.009)
scale(log(num_new_dev + 0.1))		0.922*** (0.020)
scale(num_w_tool_expos)		-0.062*** (0.009)
scale(log(num_involved_dev + 0.1))		-0.038** (0.016)
scale(num_pos_dev)		-0.030** (0.013)
scale(num_neg_dev)		-0.013 (0.012)
Constant	1.838*** (0.155)	1.644*** (0.098)
Observations	1,085	1,085
Log Likelihood	-5,868.721	-4,451.806
Akaike Inf. Crit.	11,749.440	8,925.611
Bayesian Inf. Crit.	11,779.380	8,980.494
Marginal R^2	31.50%	77.99%
Conditional R^2	86.71%	91.33%

Note:

* $p < 0.1$; ** $p < 0.05$; *** $p < 0.01$

Of the control variables only *project_age* has a significant, sizeable positive effect. It is in line with expectations: older active projects will have more participants and this likely longer discussions. In the team variables, *num_new_dev* is sizeable and positively associated with the discussion length, all else kept constant. Both DOI and SJT are consistent with these findings, as the new people, who have little ego involvement, can be the ones with questions or comments about tools and the project. The other team variables are negatively associated with the discussion length, but their effects are small. In particular, *num_involved_dev* and *num_w_tool_expos* are negatively associated with discussion length, in line with expectations that involved developers and those exposed to the tool previously may not need long discussions to decide. The *num_pos_dev* variable has a small, but a significant negative effect on discussion length, suggesting that more positive developers can be beneficial to shortening discussions.

Answer to RQ4: For adoption success, the positive significant variables are the number of mentions, developers with prior exposure, and involved developers. As for discussion length, the number of new developers seems to be the most significant indicator to extend the discussion, while the exposure factor has a positively sizable effect on shorting the discussion.

7 TAKEAWAYS FOR PRACTITIONERS

Here we distill from our findings some practical takeaways and suggestions. Generally, OSS project team discussions are helpful for community building. But they are sorely lacking during tool adoptions, and since we also found that the discussions tend to be goal-oriented and rational, we recommend that they should be encouraged in the OSS community.

Our finding that having more people on the project with prior exposure to a tool is associated with successful adoption is evidence toward proceeding with adoptions after a team has multiple members with prior exposure. However, longer discussions can be

distracting to a team, and we found they are not associated with better adoption outcomes; on the other hand, having an influencer as a champion for the adoption may help.

We also found that some tools are associated with longer discussions than others, perhaps because they demand certain prior exposure and further research. Setting expectations for the team ahead of time can limit feelings of frustrations arising out of lengthy discussions. And while we found no association between negative (or positive) developers and adoption success, discussions tend to be shorter as the number of positive developers increases. Thus, being more positive than negative may help keep things shorter.

Further, more new developers are associated with lower adoption and longer discussions. While the number of new developers cannot be modulated much, perhaps timing tool adoptions during periods of low influx of new people may help the proposed tool adoption. Finally, more commits and comments are associated with lowered success of adoption, thus, planning tool adoptions away from busy project times may result in more successful adoptions.

8 THREATS TO VALIDITY AND CONCLUSION

Threats. Both adoption data and commits/comments data were gathered from GitHub, thus, generalizing the results beyond GitHub, or even beyond the gathered corpus, carries some risk. However, the projects were selected randomly (with some minimum activity requirements), thus lowering this risk. Also, we do not consider any offline/in-person communication channels between developers except through GitHub. Previous work has found that there exists a notable decrease in communications associated with same company affiliation, implying that developers may share their opinions offline. Also, Senti4SD is trained on communications among developers in Stack Overflow. GitHub comments can be different than Stack Overflow comments, though our small sample study here capped that to 12%.

Conclusion. Motivated by the availability of multiple tools per use category in DevOps settings, and the general lack of guidance about their appropriateness for specific projects, we studied team level determinants of tool adoptions and discussions.

In terms of the relative timing of tool adoptions, we found that there is a significant difference between the distribution of adoption times for tools in the same use category, making it challenging to choose which tool to adopt for projects that are laggards.

We considered tool adoption as a project-wide phenomenon, affecting every member in the group. But also depending on the opinion of many of them, including their prior impressions of tools and linguistic sentiment, we demonstrated that the involvement, tenure, and more importantly, prior exposure to the tool play significant roles in the discussion. We also find that strong influencers are associated with more, successful adoptions.

We find that the attitude towards adoptions varies across different groups of people, and that a team's relative negativity is tool-specific, and can change after adoption, suggesting that the usability of tools can be over- and underestimated. We conducted topic analysis, and in-depth case studies on how and why some similar projects choose one tool over another one. We conclude that tool adoption is akin to a reasonable team negotiation, that proceeds through multiple phases. We hope our results can be of help in future tool adoption decisions.

ACKNOWLEDGEMENTS

We are grateful to the National Science Foundation for funding this project, under grant #1717370. We thank the ASE 2020 reviewers for their constructive comments which helped improve this paper.

REFERENCES

- [1] Mark S Ackerman, Juri Dachtera, Volkmar Pipek, and Volker Wulf. 2013. Sharing knowledge and expertise: The CSCW view of knowledge management. *Computer Supported Cooperative Work (CSCW)* 22, 4-6 (2013), 531–573.
- [2] Dane Bertram, Amy Volda, Saul Greenberg, and Robert Walker. 2010. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. 291–300.
- [3] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.
- [4] Berndt Brehmer. 1976. Social judgment theory and the analysis of interpersonal conflict. *Psychological bulletin* 83, 6 (1976), 985.
- [5] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. [Journal First] Sentiment Polarity Detection for Software Development. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 128–128.
- [6] Laurent Christophe, Reinout Stevens, Coen De Roover, and Wolfgang De Meuter. 2014. Prevalence and maintenance of automated functional tests for web applications. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 141–150.
- [7] Dan Cornell. 2012. Remediation statistics: what does fixing application vulnerabilities cost. *Proceedings of the RSAConference, San Francisco, CA, USA* (2012).
- [8] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2016. Findings from GitHub: methods, datasets and limitations. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 137–141.
- [9] Fabrício R de Souza, Augusto CSA Domingues, Pedro OS Vaz de Melo, and Antonio AF Loureiro. 2018. MOCHA: A Tool for Mobility Characterization. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, 281–288.
- [10] Fabian Fagerholm, Alejandro Sanchez Guinea, Jay Borenstein, and Jürgen Münch. 2014. Onboarding in open source projects. *IEEE Software* 31, 6 (2014), 54–61.
- [11] William R Ferrell. 1985. Combining individual judgments. In *Behavioral decision making*. Springer, 111–145.
- [12] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. 2006. A model for technology transfer in practice. *IEEE software* 23, 6 (2006), 88–95.
- [13] Sandra Graham and Shari Golan. 1991. Motivational influences on cognition: Task involvement, ego involvement, and depth of information processing. *Journal of Educational psychology* 83, 2 (1991), 187.
- [14] Sarra Habchi, Xavier Blanc, and Romain Rouvoy. 2018. On adopting linters to deal with performance concerns in android apps.
- [15] Jeffrey T Hancock, Kailyn Gee, Kevin Ciacchio, and Jennifer Mae-Hwah Lin. 2008. I'm sad you're sad: emotional contagion in CMC. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*. ACM, 295–298.
- [16] Bill C Hardgrave, Fred D Davis, and Cynthia K Riemenschneider. 2003. Investigating determinants of software developers' intentions to follow methodologies. *Journal of Management Information Systems* 20, 1 (2003), 123–151.
- [17] Michael Hilton, Jonathan Bell, and Darko Marinov. 2018. A large-scale study of test coverage evolution. In *ASE*. 53–63.
- [18] Md Rakibul Islam and Minhaz F Zibran. 2018. A comparison of software engineering domain specific sentiment analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 487–491.
- [19] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 135–146.
- [20] Brittany Johnson, Rahul Pandita, Justin Smith, Denae Ford, Sarah Elder, Emerson Murphy-Hill, Sarah Heckman, and Caitlin Sadowski. 2016. A cross-tool communication study on program analysis tool notifications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 73–84.
- [21] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 672–681.
- [22] Tiffany Brooke Jordan, Brittany Johnson, Jim Witschey, and Emerson Murphy-Hill. 2014. Designing Interventions to Persuade Software Developers to Adopt Security Tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers (Scottsdale, Arizona, USA) (SIW '14)*. ACM, New York, NY, USA, 35–38. <https://doi.org/10.1145/2663887.2663900>
- [23] Francisco Jurado and Pilar Rodriguez. 2015. Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues. *Journal of Systems and Software* 104 (2015), 82–89.
- [24] David Kavaler, Asher Trockman, Bogdan Vasilescu, and Vladimir Filkov. 2019. Tool choice matters: JavaScript quality assurance tools and usage outcomes in GitHub projects. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 476–487.
- [25] Truman Lee Kelley. 1925. The applicability of the Spearman-Brown formula for the measurement of reliability. *Journal of Educational Psychology* 16, 5 (1925), 300.
- [26] J Kottmann, G Ingersoll, J Kosin, and B Galitsky. [n.d.]. The Apache OpenNLP library.
- [27] Paul M Leonardi. 2014. Social media, knowledge sharing, and innovation: Toward a theory of communication visibility. *Information systems research* 25, 4 (2014), 796–816.
- [28] Changzhou Li, Yao Lu, Junfeng Wu, Yongrui Zhang, Zhongzhou Xia, Tianchen Wang, Dantian Yu, Xurui Chen, Peidong Liu, and Junyu Guo. 2018. LDA meets Word2Vec: a novel model for academic abstract clustering. In *Companion Proceedings of the The Web Conference 2018*. 1699–1706.
- [29] Jennifer Marlow and Laura Dabbish. 2013. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 145–156.
- [30] Samim Mirhosseini and Chris Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 84–94.
- [31] Shinichi Nakagawa and Holger Schielzeth. 2013. A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in ecology and evolution* 4, 2 (2013), 133–142.
- [32] Shari Lawrence Pfleeger. 1999. Understanding and improving technology transfer in software engineering. *Journal of Systems and Software* 47, 2-3 (1999), 111–124.
- [33] Andreas Poller, Laura Kocksch, Sven Türke, Felix Anand Epp, and Katharina Kinder-Kurlanda. 2017. Can security become a routine?: a study of organizational change in an agile software development group. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, 2489–2503.
- [34] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the content of GitHub README files. *Empirical Software Engineering* 24, 3 (2019), 1296–1327.
- [35] Cynthia K. Riemenschneider, Bill C. Hardgrave, and Fred D. Davis. 2002. Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *IEEE transactions on Software Engineering* 28, 12 (2002), 1135–1145.
- [36] Everett M Rogers. 2002. Diffusion of preventive innovations. *Addictive behaviors* 27, 6 (2002), 989–993.
- [37] John Rohrbaugh. 1979. Improving the quality of group judgment: Social judgment analysis and the Delphi technique. *Organizational Behavior and Human Performance* 24, 1 (1979), 73–92.
- [38] Jukka Ruohonen and Ville Leppänen. 2018. Toward validation of textual information retrieval techniques for software weaknesses. In *International Conference on Database and Expert Systems Applications*. Springer, 265–277.
- [39] Farhana Sarker, Bogdan Vasilescu, Kelly Blincoe, and Vladimir Filkov. 2019. Socio-technical work-rate increase associates with changes in work patterns in online projects. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 936–947.
- [40] Leif-Gerrit Singer. 2013. *Improving the adoption of software engineering practices through persuasive interventions*. Lulu.com.
- [41] Philippe Skolka, Cristian-Alexandru Staicu, and Michael Pradel. 2019. Anything to Hide? Studying Minified and Obfuscated Code in the Web. In *The World Wide Web Conference*. ACM, 1735–1746.
- [42] Stuart Soroka, Patrick Fournier, and Lilach Nir. 2019. Cross-national evidence of a negativity bias in psychophysiological reactions to news. *Proceedings of the National Academy of Sciences* 116, 38 (2019), 18888–18892.
- [43] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M German. 2016. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering* 43, 2 (2016), 185–204.
- [44] Chandrasekar Subramaniam, Ravi Sen, and Matthew L Nelson. 2009. Determinants of open source software project success: A longitudinal study. *Decision Support Systems* 46, 2 (2009), 576–585.
- [45] Jiratep Tantisuwanakul, Yusuf Sulistyo Nugroho, Raula Gaikovina Kula, Hideaki Hata, Arnon Rungsawang, Pattara Leelapruete, and Kenichi Matsumoto. 2019. A topological analysis of communication channels for knowledge sharing in contemporary GitHub projects. *Journal of Systems and Software* 158 (2019), 110416.
- [46] Leigh Thompson and Terri DeHarpport. 1994. Social judgment, feedback, and interpersonal learning in negotiation. *Organizational Behavior and Human Decision Processes* 58, 3 (1994), 327–345.

- [47] Kristín Fjólá Tómasdóttir, Mauricio Aniche, and Arie van Deursen. 2017. Why and how JavaScript developers use linters. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 578–589.
- [48] 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 *Proceedings of the 40th International Conference on Software Engineering*. ACM, 511–522.
- [49] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*. IEEE, 188–195.
- [50] Georg Von Krogh, Sebastian Spaeth, and Karim R Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research policy* 32, 7 (2003), 1217–1241.
- [51] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 260–271.
- [52] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social influences on secure development tool adoption: why security tools spread. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 1095–1106.
- [53] Marvin V Zelkowitz. 1995. Assessing software engineering technology transfer within NASA. *NASA technical report NASA-RPT-003095*. National Aeronautics and Space Administration, Washington, DC (1995).
- [54] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 871–882.