Heard It through the Gitvine:
An Empirical Study of Tool Diffusion across the *npm* Ecosystem

Hemank Lamba, Asher Trockman, Daniel Armanios, Christian Kästner, Heather Miller, Bogdan Vasilescu

{hlamba, ashert}@cs.cmu.edu, {darmanios, heathermiller, vasilescu}@cmu.edu

Carnegie Mellon University, USA

ABSTRACT

Automation tools have become essential in contemporary software development. Tools like continuous integration services, code coverage reporters, style checkers, dependency managers, etc. are all known to benefit developer productivity and software quality. Some of these tools are widespread, others are not. How do these automation "best practices" spread? And how might we facilitate the diffusion process for those that have seen slower adoption? In this paper, we rely on a recent innovation in transparency on code hosting platforms like GitHub—the use of repository badges—to track how automation tools spread in open-source ecosystems through different social and technical mechanisms over time. Using a large longitudinal data set, network science techniques, and survival analysis, we study which socio-technical factors can best explain the observed diffusion process of a number of popular automation tools. Our results show that factors such as social exposure, competition, and observability affect the adoption of tools significantly, and they provide a roadmap for software engineers and researchers seeking to propagate best practices and tools.

CCS CONCEPTS

• **Software and its engineering** → *Software libraries and repositories*; • **Human-centered computing** → **Empirical studies in collaborative and social computing**.

KEYWORDS

software tools, innovations, diffusion, open source ecosystem

1 INTRODUCTION

Tools – from mundane ones like email to more specialized ones like build managers, continuous integration services, and static analyzers – are so integral to software engineering that for many it is unimaginable to live, and develop software, without them. This
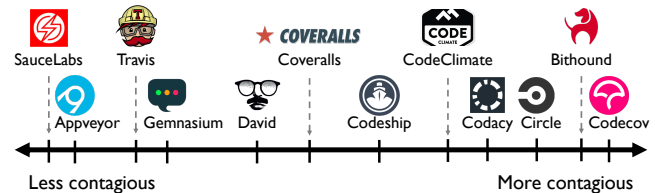
Figure 1: There is variance in the contagiousness of the different tools in our study (details in Section 5).

is especially the case nowadays, with the growing popularity of the DevOps movement centered around automation of processes and practices, and the myriad of tools implementing this automation. Most software tools are useful, enabling improvements in developer productivity and software quality. But not even the most handy of tools are used universally, or spread equally fast among practitioners once released. For example, in the *npm* open-source ecosystem, we have observed variance even among popular quality assurance tools in how "contagious" the different tools are (Fig. 1), *i.e.*, how quickly they spread and get adopted by different projects.

Prior software engineering work offers one possible explanation why some tools are less adopted: simply becoming aware of tools that might be useful is a challenge [52, 53, 84]. More broadly, Rogers' well-established theory of innovation diffusion [61] identifies five principal attributes of innovations (*e.g.*, tools) that influence their adoption: their observability, relative advantage, compatibility, complexity, and trialability. For example, the theory predicts that the more observable (*i.e.*, visible) and the more trialable (*i.e.*, available for experimentation) the innovation is, the more likely an individual becomes to adopt it [61, 68] (more details in Sec. 2).

On "social coding" platforms like GitHub, GitLab, and Bit-Bucket observability is high. Such platforms offer a high level of transparency [16, 17], a wealth of channels through which developers and software projects can become interconnected [45], and a diversity of signals [41, 73] through which people can get exposed to each other's practices and tools. For example, for GitHub open-source projects using continuous integration, the pass or fail status of each build is automatically displayed alongside each commit in the user interface as cross (✗) or check signs (✓), which link to a specific tool's website (*e.g.*, Travis, Circle) and the build logs produced by that tool; project maintainers can also choose to display additional repository badges (*e.g.*, `build passing`, `build failing`) indicating the continuous integration build outcome in real time, also linking to the specific tool and corresponding build log.

What's the tipping point? How many developers must be exposed to a new tool or technology, and under what conditions, before it catches on and is adopted en masse? What channels or other means of signaling adoption influence tool diffusion the

most? And what explains why some people become early adopters while others adopt late or not at all? While there is prior software engineering literature on the diffusion of software development methodologies [32, 65], object orientation [22], programming languages [2, 6, 47], and security tools [82, 84], quantitative, data-driven answers to the questions above are hard to come by. Quantitatively understanding and modeling the complex mechanisms through which software tools diffuse within practitioner communities can be beneficial: for example, one can begin to design interventions to promote adoption of beneficial tools and practices *based on empirical evidence* rather than just intuition, perceptions, or beliefs; the former don't always agree with the latter [19].

In this paper we present a novel methodology to begin to quantitatively answer the questions above, that combines the creation of a large-scale longitudinal network dataset of tool adoptions (Sec. 5) with statistical hazard modeling techniques (Sec. 6). Our key insight that allows us to study data on adoption at scale, across many tools and projects, is that *repository badges*, *i.e.*, those images such as `build passing`, `dependencies up to date`, and `coverage 94%` embedded into a project's README file, are readily observable and consistent across many tools, thus they can be used as proxies indicating usage of the underlying software tools, *i.e.*, continuous integration, dependency management, and code coverage in these specific examples. Moreover, since the README file is part of a project's version control system, one can also precisely infer when a project adopted each tool.

Using this methodology and grounded in sociological theory (Sec. 2), we report on a multiple case study on the diffusion of the 12 most popular cloud-based *automated testing* and *continuous integration services*, *code coverage reporters*, and *dependency managers* used within the *npm* ecosystem on GitHub (Table 1). Specifically, we test hypotheses about factors theorized to impact how rapidly the software quality assurance tools spread through a heterogeneous network, focusing on the tools' observability, relative advantage, compatibility, and complexity (Sec. 3). Notably, our methodology enables two refinements not typically present in prior work. First, we provide estimates of the relative strength of the effects of different social and technical network ties between projects, through which developers can observe each other's practices and tools. Second, we estimate the effects of these observability-related factors while accounting for theorized covariates, such as competition between tools and degree of context fit between tools and project environments.

At a high level, our results (Sec. 7) confirm the extent to which observability plays a key role in software tool adoption, as the theory predicts, and reveal that both social and technical network ties between projects contribute to explaining tool adoption (in addition to other theorized factors). However, while multiple types of ties matter, the social ties introduced by shared committers have the strongest effects on tool adoption, more so than, *e.g.*, ties introduced by direct technical dependencies. These results have implications for software engineering researchers and tool builders (Sec. 8). For researchers studying tool adoption, we provide a robust and theoretically grounded framework to study the diffusion of other tools and practices, including the dataset compiled as part of our study as a possible starting point. For tool builders and platform designers, we provide evidence-based suggestions on where investments

Table 1: Overview of the software quality assurance tools in our study, organized by task equivalence class; see Sec. 5.

Class	Tools
Cont. integration	TRAVIS, CIRCLE, APPVEYOR, CODESHIP
Dependency mgmt	DAVID, BITHOUND, GEMNASIUM
Code coverage	COVERALLS, CODECLIMATE, CODECOV, CODACY
Cross browser test.	SAUCELABS
Control group	SLACK, GITTER, LICENSE, NPM-VERSION, NPM-DOWNLOADS

to catalyze the adoption of software engineering tools and best practices might pay off the most, given similar networks.

In summary, we contribute: (1) a longitudinal dataset of tool adoptions for 12 quality assurance tools across a heterogeneous network of *npm* open-source packages; (2) a methodology and in-depth analysis (using hazard modeling) of the effects of multiple socio-technical factors on the adoption of each tool, testing hypotheses grounded in sociological theory; (3) a characterization of early tool adopters; and (4) a discussion of results with implications for researchers and tool builders.

2 THEORY AND RELATED WORK

Our work connects general social science *theories* of adoption of innovations to software engineering *practice* in transparent social coding environments like GITHUB, GITLAB, and BITBUCKET, where the potential to become exposed to new tools and practices is particularly high, as we will argue. In this section we review prior work and relevant theories, before we derive testable hypotheses in Section 3.

Adoption of software tools. Software engineering researchers and practitioners are often frustrated by the low adoption of practices and tools, such as static analysis, refactoring tools, program comprehension, or security testing, even though many of them are generally perceived to be beneficial. Researchers have found a wide range of technical and social pain points in tool adoption, such as false positives in static analysis tools [e.g., 39, 64, 71], missing trust in correctness [e.g., 74], crypting tool messages [e.g., 38, 39], slow response times [e.g., 80], lack of workflow integration [e.g., 36, 41, 64, 86], lack of collaboration support [39], lack of management buy-in [e.g., 18, 80], overwhelming configuration effort [e.g., 25, 36, 39, 71, 80], and simply a lack of knowledge about tools [e.g., 60, 74, 87]. In response, most software engineering research has focused on technical solutions, such as improving functionality, accuracy, and performance [e.g., 8, 64, 72], improving usability [e.g., 38, 46, 51, 72, 79], and improving discoverability through recommendation mechanisms or process integration [e.g., 49, 64, 87]. In this work, we explore an orthogonal question that does not focus on a tool's functionality but on how developers learn about it and adopt it, that is, how it diffuses in a community.

Diffusion of innovations. Our main theoretical framework is Rogers' [61] well-established diffusion of innovations theory, consisting of five main *elements*: (a) the innovation itself, (b) the adopters, (c) the communication channel, (d) time, and (e) the social system. The *innovation* can be any idea or practice that is novel to a group or individual. *Adopters* are any individuals or entities that make the decision to adopt the innovation or not. The *communication*

channels are the media through which innovations diffuse. *Time* is inherently necessary for innovations to spread; not all individuals will adopt the innovation at the same rate. The *social system* can be considered as a combination of contextual, cultural, and environment influences, *e.g.*, external influences such as mass media and internal influences such as opinion leaders.

The theory also describes five *attributes* of innovations that influence their adoption: observability, relative advantage, compatibility (congruence with the individual's context), complexity, and trialability (availability for experimentation). In short, the more observable, advantageous, compatible, and trialable, and the less complex an innovation is, the more likely it is to get adopted.

Roger's theory has been applied to a diversity of innovations across a variety of domains, *e.g.*, new types of corn by Iowa farmers [63], family planning practices [62], mathematics in schools [12], new medical technology [4], and language [59].

Diffusion of software innovations. Rogers' theory has also been used in the software engineering and programming languages communities, to inspire or inform the design of adoption studies of software development methodologies [13, 32, 34, 40, 65], object orientation [22], programming languages [2, 6, 47], and security tools [82, 84]. While most software engineering research on tool adoption has focused on technical and social issues (as discussed above), most factors identified by prior studies to influence the adoption of software-related innovations can generally be traced back to the elements of Rogers' theory. Let us highlight three examples, closest to our work: (1) Xiao *et al.* [84] studied how security tools (the innovation) spread within commercial software companies (the social system). Based on an interview study with 42 professional software developers, the authors found that "developers are more likely to adopt tools they learn about from their peers than ones they learn about elsewhere," and that "developers who interact with security teams as part of the auditing process are more likely to adopt security tools," two findings which can be traced back to *observability* and *trialability*. (2) In a follow-up study, Witschey *et al.* [82] surveyed developers from 14 companies and 5 mailing lists about their use of security tools and quantified the relative importance of self-reported tool adoption factors. The authors found that the strongest predictor of security tool use in a multiple logistic regression was "developers' ability to observe their peers using security tools," further validating empirically the importance of *observability*. (3) Finally, Rahman *et al.* [58] surveyed 268 software professionals about their adoption of another innovation—build automation tools—finding that *compatibility* with the practitioners' existing tools and workflows is the strongest predictors of tool use among the different self-reported factors.

Framing of our work. We consider open-source software development on social coding platforms like GitHub as the *social systems* in which the innovations spread, and open-source projects part of the *npm* ecosystem as the *adopters*. The specific *innovations* we consider are a range of popular GitHub-integrated quality assurance tools (Table 1). By design, all the innovations we consider are expected to have similar *trialability* (they are all freely available online and tightly integrated with GitHub). Our study investigates how the other four important attributes of innovations impact their

diffusion: their *observability* through different *communication channels* on GitHub, their relative *advantage*, their *compatibility* with the adopter's context, and their relative *complexity* compared to other practices with lower expected adoption cost.

According to Rogers' theory, the decision to adopt an innovation proceeds in stages, starting with becoming aware of the innovation, building familiarity, making a decision and acting on it, and finally re-evaluating whether to continue using the innovation or not [68]. Our work addresses primarily the earlier stages of gathering and processing information about the innovation, to form perceptions which later inform a decision to adopt or reject [1]. This is when the *observability* of an innovation is expected to have most impact.

While the three related studies discussed above and our current work all share the same goal—providing stakeholders with empirically-validated, evidence-based recommendations to promote the adoption of beneficial software tools and practices—our work is novel in several key ways. First, we adopt a *network science perspective* and dive deep into the different channels through which practitioners in GitHub-like transparent environments can become aware of new tools and practices, quantifying the relative impact of each channel on adoption. Second, we use sound hazard modeling techniques to analyze multidimensional longitudinal *trace data* mined from thousands of open-source projects, therefore our work provides robust triangulation for results from prior qualitative studies; our observational study is robust to the response biases, recall errors, and self-serving biases typical of surveys and interviews. As a side effect, our work also helps to generalize prior findings to developers outside of industry.

3 DEVELOPMENT OF HYPOTHESES

Building awareness of potentially useful tools is a known challenge for software developers [30, 52]. Social coding platforms like GitHub provide opportunities for tool use to be observed among projects hosted there due to high levels of transparency [16, 17]. A wealth of *signals*, such as traces in log files, status checks for commits,[1] and repository badges [73], are all publicly and prominently visible in the user interface, providing high levels of transparency into the tools and practices used in those projects. There is a diversity of channels through which open-source developers can be exposed to these signals. Channels through which influence can flow on platforms like GitHub include collaboration in the same repositories, pull requests, and follower relationships, all of which introduce links between adopters.

If heightened visibility is a crucial condition to diffusion, then what drives an individual's limited attention towards new tools and technologies is a crucial factor to adoption [55]. The sociology literature offers mechanisms that compete to prompt the attention needed for an innovation's adoption, and describe which groups of users are more susceptible to adopt. From the literature, we select hypotheses that can be operationalized using empirical data that is available, and only comment on them. The following hypotheses in one way or another all help in understanding the mechanisms through which tools diffuse through a highly transparent and networked environment like the *npm* ecosystem on GitHub:

---

[1]https://help.github.com/en/articles/about-status-checks

Early adopters as a driver of diffusion. The early adopters of an innovation have arguably the highest impact on the eventual spread of the innovation. According to Rogers' theory, early adopters are more socially forward than other groups, and they enjoy higher social status and influence [68]; in our context, they may also be more willing to experiment with new technologies and to take risks. Due to their openness, opinion leadership, and high social status, early adopters would be the ideal target for interventions to increase tool use. it is they who should be targeted first by "change agents" [61], since they set the adoption pace. We hypothesize:

H_1. *The higher the social standing and technological openness of a project maintainer, the more likely the project is to adopt a new tool.*

Network ties as a driver of diffusion. From a network theory perspective, ties between individuals in a network are what drives diffusion: Ties to peers can help individuals identify the most promising ideas [11], as well as help build the necessary knowledge to execute these ideas [31]. While there is a debate as to whether such influence is indirect through shared common affiliations that attract individuals together (*e.g.*, homophily) or more direct learning between individuals that are idiosyncratic to the relationship and not solely based on shared affiliations (*e.g.*, peer influence), the network perspective argues that diffusion and adoption occur through interaction amongst tied individuals [5, 54]; network ties are what influences an individual's limited attention towards which practices they should adopt or not. To the case here, network theorists could argue that the diffusion of a specific tool is based on the prior experiences that developers bring to their current software project. If a developer on a project has worked on a prior project that adopted a tool, that coder is likely to bring that practice to their current project through this *social connection*. We hypothesize:

H_2. *The stronger the social connection of a project to other projects that adopted a tool, the more likely the project is to adopt that tool.*

Resource dependencies as a driver of diffusion. The resource dependency perspective argues that adoption of a practice is driven by power asymmetries between those who hold a critical resource and those who depend on it: The former can dictate the activities and operations of their dependents [57], and the latter must continuously gather information so as to be attuned to what that key resource holder does and maintain or adapt their operations accordingly [24, 56]. This perspective argues that critical resource holders are what influences an individual's limited attention towards which practices they should adopt or not. Resource dependency theorists could argue that the diffusion of a specific tool will depend upon what critical resource holders adopt. Thus, if a project has key *technical connections* to other projects (*e.g.*, through library reuse), it is likely to pay attention to those who manage these key dependencies and what they do, to adapt accordingly if needed. We hypothesize:

H_3. *The stronger the technical connection of a project to other projects that adopted a tool, the more likely the project is to adopt that tool.*

Legitimacy as a driver of diffusion. The institutional theory perspective argues that what drives diffusion of a "best practice" is whether it is deemed as legitimate. Legitimacy is defined as "a generalized perception or assumption that the actions of an entity are desirable, proper, or appropriate within some socially constructed system of norms, values, beliefs, and definitions" [69]. From this perspective, adopting a practice happens through mimicry, copying what others seem to already accept as legitimate [21]. In this process, the individual first looks to their environment for legitimate models of how they ought to understand their social world of interest, and then adopt practices conforming to that model [66]. In our case, institutionalists could argue that the more some tools become widely-adopted by other projects in the social system (*i.e.*, all of GitHub), the more legitimate one's project will be seen by other developers if it were to also adopt those tools. We hypothesize:

H_4. *The more widely adopted tools are globally, the more likely a project is to also adopt those tools.*

Functional need as a driver of diffusion. While the previous hypotheses all describe plausible tool diffusion mechanisms, it is important to acknowledge that software projects span a diversity of sizes, application domains, lifecycle stages, *etc* and, as a result, their tool needs are likely to differ. For example, a continuous integration service and a code coverage reporter would be an unusual choice for projects without an automated test suite. Therefore, one has to study tool adoption behavior considering the software projects' underlying need for, and compatibility with, different tools. In line with Rogers' theory, we hypothesize that tool compatibility or congruence with the adopter's context is a main driver of diffusion. Stated differently, if past adopters of a tool reflect some underlying need for that tool among projects of that type:

H_5. *The more functionally similar a project is to other projects that adopted a tool, the more likely the project is to adopt that tool.*

Market forces as a driver of diffusion. Software developers nowadays often have a range of competing tools to choose from for any given task. For example, one could choose between Travis, Circle, and Appveyor for their continuous integration needs on GitHub, just to name a few. At the same time, developers rarely choose more than one competing tool for a given task, and rarely switch tools once they have made a choice [41, 80]. Institutionalists also note that getting an additional signal for the same source is seen as redundant and has diminishing returns [44]. We hypothesize:

H_6. *Adopting a tool from an equivalence class makes it less likely to adopt a competing tool from the same class.*

Complexity as a hindrance of diffusion. All the software quality assurance tools we consider in this study (cloud-based automated testing and continuous integration services, code coverage reporters, and dependency managers) involve some kind of underlying analyses (*e.g.*, of builds and tests failing, or of dependencies being up-to-date). They have relatively higher complexity and require relatively more effort to adopt than other types of tools, *e.g.*, communication tools like Slack and Gitter. For example, adopting continuous integration requires at the very least setting up a build system and an automated test suite, which is non trivial. Therefore, we hypothesize that quality assurance tools are slower to diffuse and require more social influence to spread because of the extra effort to adopt:

H_7. *Quality assurance tools require more social influence to diffuse than tools without underlying analyses.*

4 STUDY OVERVIEW

We first describe our study design at a high level, before diving into the details of each step. Our key idea, inspired by Trockman et al. [73], is to estimate the adoption of the different cloud-based automated testing and continuous integration services, code coverage reporters, and dependency managers across all *npm* package repositories on GitHub (over half a million at the time of data collection) by tracking the spread of the corresponding repository badges, *e.g.*, `build passing`, `coverage 94%`, and `dependencies up to date`; using the badges implies using the underlying tools.[2]

We chose GitHub as it is currently the most popular software development platform, with a broad range of tightly-integrated and widely-used software quality assurance tools. We chose *npm* as it is currently one of the largest and most popular open-source ecosystems; by studying the diffusion of different tools within a fixed software ecosystem we can reduce the influence of external environmental and contextual confounding factors.

There are multiple reasons for studying badges as proxies for tool use, instead of artifacts such as configuration files. Most importantly, badges are *easily observable* signals, often more readily observable and easier to interpret than changes to configuration files. A developer interacting with a software project on a platform like GitHub would likely notice badges first, before noticing tool-specific configuration files. This makes badges more relevant than configuration files for studying social influence in tool adoption. Second, from a pragmatic study design perspective, we use badges as a *uniform proxy* for tool use. That is, we can detect with high precision (and given current practices often reasonably high recall) when projects use certain tools. The detection is much easier, scales much better (analyzing README files rather than all files in a repository), and is much more uniform across tools (which may be configured in one of many different files, *e.g.*, depending on which build tool or CI service is used, and sometimes without leaving any publicly visible traces in the repository itself). Note that some developers may perceive badges as increasingly noisy,[3] possibly diminishing their value as signals somewhat. Our study is neutral on this issue but we do acknowledge this fundamental limitation of our design.

Next, to track how badges (tools) *spread*, we construct a heterogeneous longitudinal *network*, in which nodes represent *npm* repositories and edges represent different interconnections between repositories corresponding to our hypotheses (e.g., dependency relation, joint contributors, project similarity). For each node, we track *when* they adopt each badge; for each edge, we track *when* the edge was introduced.

We then operationalize various characteristics of projects and developers, and use multivariate statistical modeling on this network to test our hypotheses. Specifically, we use *logistic regression* to model the characteristics of early badge adopters to test H_1 (recall, this is the only developer-level hypothesis in our study; the others are project-level). For hypotheses H_2–H_6, we analyze the network's adoption history using *survival analysis*, to model and

Table 2: Overview statistics for our dataset.

Repositories	168,510
Commits	14,686,752
Developers (committers, pull req. submitters, watchers)	871,089
Repositories that adopted at least one badge	86,768

compare which of the different diffusion mechanisms above associate with faster adoption through the network. In this step we build separate survival models for each of the tools we consider.

Finally, for H_7 we qualitatively[4] compare the coefficient estimates for the social-diffusion-related variables in our models between the main quality assurance tools and the control group tools / badges. The 12 quality assurance tools we consider all have non-trivial underlying analyses/executions and all require non-trivial setup, *i.e.*, they are at least somewhat complex to use. Per the theory, this complexity should act as a hindrance to adoption. Therefore, when compared to a "control group" of tools with relatively much simpler setup/usage and no underlying analyses (see Table 1), we expect the 12 quality assurance tools to require more social influence to diffuse. In our models, if the distribution of estimated social influence model coefficients / hazard ratios for the experimental group appears higher than for the control group, we would consider H_7 supported.

5 DATA COLLECTION

We start from the longitudinal dataset collected by Trockman et al. [73], containing 92 badges adopted by different packages across *npm*. This dataset was collected from GHTorrent [28], the GitHub API, and `git` logs; it contains all *npm* packages which could be cross-linked to their corresponding GitHub repositories at the time of data collection in mid 2017. We then focus on a subset of tools, perform several data cleaning steps, and compute additional network data to test our hypotheses. This section describes our data collection steps.

5.1 Tool Selection

From the original dataset, we select the badges corresponding to the most popular 12 quality assurance tools, ranging from the continuous integration tool Travis as the most popular (60, 350 adopters) to another continuous integration tool Codeship as the least popular (430 adopters); see Table 1 for the complete list. For comparison when testing H_7, we also select 5 popular badges without underlying analyses (*i.e.*, with relatively lower adoption cost), as a "control group" (also listed in Table 1).

Note that the decision to focus on the most widely adopted tools, driven by the computational requirements of our data analysis technique (Section 6), may limit the generalizability of our results to relatively unpopular and newly released tools with few users. However, there is still high variance in adoption between the most popular (Travis) and least popular (Codeship) tools in our sample, and our results are generally robust to this variance, as we will show below.

---

[2]The converse is not true, one can use these tools without displaying badges. Therefore, badge adoptions represent a lower bound on the adoption of the underlying tools.
[3]For example, https://github.com/BraveUX/for-the-badge.

[4]The relatively small size of our two samples, 12 experimental tools and 5 control group tools/badges, precludes sound statistical comparison.
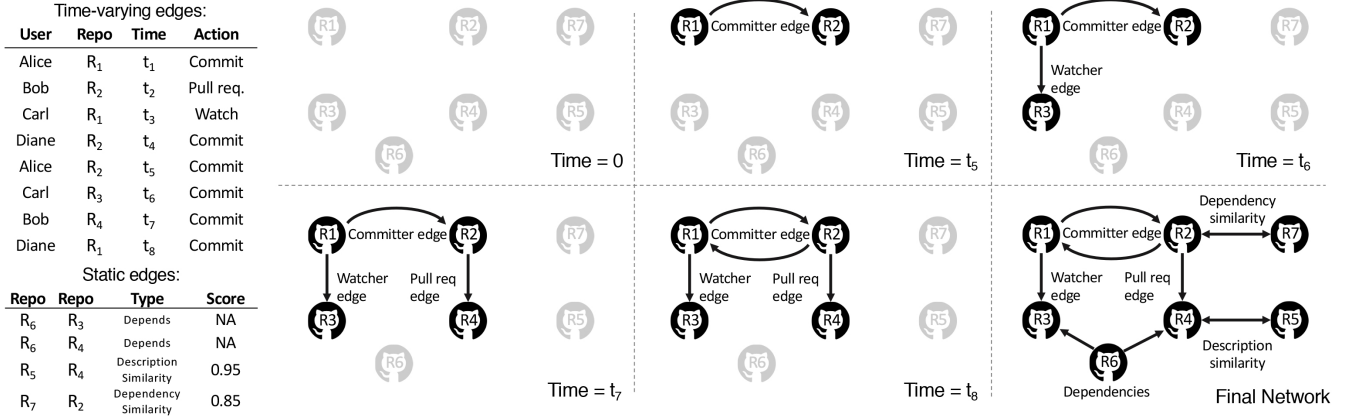
Time-varying edges:			
User	Repo	Time	Action
Alice	R_1	t_1	Commit
Bob	R_2	t_2	Pull req.
Carl	R_1	t_3	Watch
Diane	R_2	t_4	Commit
Alice	R_2	t_5	Commit
Carl	R_3	t_6	Commit
Bob	R_4	t_7	Commit
Diane	R_1	t_8	Commit

Static edges:			
Repo	Repo	Type	Score
R_6	R_3	Depends	NA
R_6	R_4	Depends	NA
R_5	R_4	Description Similarity	0.95
R_7	R_2	Dependency Similarity	0.85

Figure 2: Overview of the network construction process.

5.2 Preprocessing

Data cleaning. We take several steps to further clean the original badge adoption dataset by Trockman et al. [73]. First, we filtered out small inactive repositories, which add unnecessary effort to our network data collection effort (Section 5.3), without carrying much information. As a proxy for small and inactive, we removed repositories with fewer than 10 commits recorded in GHTorrent.

Second, we removed obvious bots, which would introduce artificial links between repositories, biasing our network measures. Based on the intuition that bots are likely to have "contributed" to many repositories or authored many commits [20], we manually inspected the top 30 most active GitHub user accounts in our dataset along these two dimensions. This way we found, and subsequently removed, many instances of bots, *e.g.*, web-flow, renovate-bot.

Third, we cleaned tool adoption dates in our dataset, since they are important for our analyses. By plotting and inspecting the histograms of adoption dates per badge, we discovered sets of highly synchronized repositories, typically all owned by the same account (*e.g.*, the prefixes npmdoc/node-npmdoc-%, npmtest/node-npmtest-%). Badges in these repositories appeared automatically introduced, perhaps by a bot, because they had identical timestamps. We removed these clusters completely.

Fourth, when inspecting the data we also discovered a few repositories with identical name and creation date, that had multiple entries in GHTorrent. We combined all such repositories into one single instance and aggregated their metrics (*e.g.*, we computed the union of their respective sets of commits).

Finally, we performed de-aliasing on the developers in our sample. Multiple aliases belonging to the same individual are a well-known type of noise when mining software repositories [3, 43, 81], which can also directly impact our network measures and downstream analyses. To merge aliases, we refine an existing heuristic approach, matching on name, email address, and other similar heuristics, used in different variations in prior work [e.g. 76, 77, 81]. Our flavor of the algorithm (see Supplementary Material) makes two passes over the input data, with separate de-aliasing at project- and global-level. At project level, we tune the heuristics for high recall, since the risk of collisions (false positive matches) is smaller

because the size of the input is smaller. At global level, we use only high-precision heuristics, since the risk of collisions increases. To combine the results of the two steps, we then find connected components on the graph formed out of the merges discovered earlier (nodes are aliases, edges are links discovered during either pass). This way, each connected component represents all of a user's aliases; summarizing the distribution of the number of aliases per cluster, we observe a minimum of 1, maximum of 21, mean of 1.19, and median of 1. We validated our approach by manually analyzing a random sample of 100 alias clusters with at least two aliases. One author checked GitHub user and project contributor pages, repository git logs, and Google search results, discovering no obvious false positives; note, however, that there may still be false negatives.

Our final dataset (Table 2), a strict subset of the one by Trockman et al. [73] in terms of projects, contains 168, 510 repositories. All the network data and other variables included in our analysis are computed after de-aliasing users and, where applicable, they represent aggregations of data from individual aliases.

5.3 Network Construction

The central part of our study is the analysis of diffusion mechanisms through a network of projects: new projects becoming aware of and adopting tools already in use in other related projects. To this end, we first build graphs where nodes are repositories and edges capture different socio-technical relationships between them. Then, we combine all these graphs into a unified heterogeneous network. Finally, to enable longitudinal analysis, we update these graphs in 4-week intervals to reflect additional relationships (*e.g.*, a dependency is introduced or a developer joins a second project) and we track (based on the badges dataset above) which repositories have adopted which tool by the end of that interval. This section describes the network construction process; see also Figure 2. Note that we assume that once a link between two projects is formed, *e.g.*, through a shared contributor, that connection persists; *e.g.*, we assume that a developer leaving a project may return in the future. This means that we only add edges to our network over time, not remove them. Other operationalizations, *e.g.*, to decay edges over time, are beyond the scope here but may increase precision.

Table 3: Basic statistics for the social connection graphs.

Graph	Committer ($\mathcal{G}_{CN}$)	Watcher ($\mathcal{G}_{WN}$)
Num. nodes	148,582	150,857
Num. edges	4,461,796	16,001,804
Avg. degree	30.03	106.01
Avg. cluster. coeff.	0.80	0.36

Committer graph (social). Developers committing to multiple repositories can create strong social connections between the repositories, through which practices can spread (recall $\mathbf{H}_2$ above). For our committer graph ($\mathcal{G}_{CN}$), we add edges to represent links through developers that *authored commits to both repositories*, either directly or through pull requests. To accurately reflect the temporal order of commit events and to capture the direction of the possible influence flow, these edges are directed from the repository to which the developer committed first. Furthermore, the edge is added in the time interval in which the developer has first committed to both projects. In Figure 2 we show an example, where Alice commits to R_1 at time t_1 and then to R_2 at time t_5, resulting in an edge $R_1 \rightarrow R_2$, available at t_5 and later. We then compute, and model in our regressions below, the variable `Num_activated_`$\mathcal{G}_{CN}$`_neighbors`, counting the number of a node's neighbors in $\mathcal{G}_{CN}$ that have already adopted a given tool by a particular time window.

Watcher graph (social). Another potential information flow channel is the watcher–committer relationship: A user subscribed to a particular repository ("watching" in GitHub lingo) may observe that repository using a tool and may take that practice with them ($\mathbf{H}_2$). We add an edge (in the watcher graph $\mathcal{G}_{WN}$) when a commit author in a repository has been watching another repository, again directed from the watched repository to the one contributed to. In our example, Carl starts watching R_2 at time t_3, and then starts committing to R_3 at time t_6, thus creating the edge $R_2 \rightarrow R_3$ starting t_6. The corresponding variable in our models, computed analogously, is `Num_activated_`$\mathcal{G}_{WN}$`_neighbors`.

Note how the two social graphs, $\mathcal{G}_{CN}$ and $\mathcal{G}_{WN}$, capture different ends of the social connection spectrum: $\mathcal{G}_{CN}$ is expected to indicate the strongest connection between repositories, and $\mathcal{G}_{WN}$ the weakest, because of the relatively high bar to committing as opposed to watching. Table 3 summarizes the two graphs.

Dependency graph (technical). To capture library reuse as a technical channel of potential information diffusion ($\mathbf{H}_3$), *i.e.*, *npm* packages reusing other *npm* packages as libraries, we also extract a repository–repository dependency graph $\mathcal{G}_{dep}$ from the corresponding `package.json` files. For example, in Figure 2, R_6 is directly dependent on both R_3 and R_4. The corresponding variable in our models is `Num_activated_`$\mathcal{G}_{dep}$`_neighbors`. Note that to reduce the data collection effort, our dependency graph is static, reflecting the dependencies at the time of data collection. By manually inspecting a small sample of projects, we observed that dependency changes typically occur when new libraries get added, while existing libraries rarely get removed [85]; therefore, our static graph can be considered as an over-approximation of the true dependencies.

5.4 Other Operationalizations

Next we describe how we operationalized the other concepts involved in the hypotheses in Section 3, and several control variables.

We discuss operationalizations related to early adopters (people) first, corresponding to $\mathbf{H}_1$—recall the difference in study design for testing this hypothesis, discussed in Section 4—followed by operationalizations related to repositories, corresponding to $\mathbf{H}_4$–$\mathbf{H}_6$.

Early adopters. We used the `git` logs to identify the developers (project maintainers) who introduced each badge in each repository (recall, badges are embedded into README files, which are part of the version control system). We then collected data using GHTorrent to characterize the developers' social standing and technological openness ($\mathbf{H}_1$). We operationalize *social standing* with two variables for each developer: the total number of stars on all their projects, and the number of users that follow them. We operationalize *technological openness* in terms of programming language diversity, *i.e.*, the number of distinct languages (as determined by GitHub's `linguist` library) used across all repositories contributed to by that developer. Early adopters may also be more experienced and embedded in the open-source community, so we include a developer's total commit count and days of membership on GitHub as control variables.

But what constitutes an *early adopter*? The theory (Section 2) predicts that adopters fall into one of five categories: innovators, early adopters, early majority, late majority, and laggards. Since the *diffusion effect* typically results in a bell-shaped curve of new adopters per unit time [61], Rogers argued that these adopter categories should be defined by the standard deviations of the normal adoption curve, and that it is useful for empirical frameworks to define these so-called "ideal types" [61]. Therefore, to categorize each developer in our sample, we found the date of their first badge adoption (of any kind) across all repositories, and labeled the first 16%[5] of developers as innovators and early adopters. We will later compare the characteristics of this group to the group of early majority, late majority, and laggards (the remaining 84%).

Dependency similarity. As a proxy for functional similarity between two repositories ($\mathbf{H}_5$), we compute the degree to which their sets of dependencies are similar [70]. Specifically, we consider each set of dependencies as a "document" and compute the TF-IDF correlation matrix similarity score of the two documents [10]. We then record an undirected edge between two repositories in the $\mathcal{G}_{depsim}$ graph, *e.g.*, $R_2 \leftrightarrow R_7$ in Figure 2, if each repository has three or more dependencies (to avoid spurious links), and if their TF-IDF similarity score is 0.9 or greater; we determined the threshold empirically after manually reviewing a random sample of pairs of repositories and their similarity scores. Table 4 shows examples of pairs of repositories we discovered to be similar this way. The corresponding variable in our models, `Num_activated_`$\mathcal{G}_{depsim}$`_neighbors`, captures the number of neighboring repositories that adopted the tool in a previous time window.

Description similarity. As another proxy for functional similarity between two repositories ($\mathbf{H}_5$), we compute the degree to which their README files are similar. Specifically, we consider each README as a "document" and compute the TF-IDF similarity scores of the two documents. If the score is 0.7 or greater (threshold chosen empirically, as above), we record an undirected edge between the two repositories in the network above, *e.g.*, $R_4 \leftrightarrow R_5$

---

[5]In a normal distribution values between the 16th and the 84th percentile ranks, *i.e.*, within one standard deviation of the mean, are considered "within the normal range."

Table 4: Examples of dependency similarity scores. Common dependencies are marked in bold.

Repo 1	Repo 2	Score
bluedapp/rotate-log: **winstonjs/winston, substack/node-mkdirp, winstonjs/winston-daily-rotate-file**	treelite/asuka: **winstonjs/winston, substack/mkdirp, winstonjs/winston-daily-rotate-file**, tj/commander.js	0.95
tandrewnichols/grunt-only: **chalk/chalk, caolan/async, lodash/lodash**	shama/grunt-hub: **chalk/chalk, caolan/async, lodash/lodash**	1.0

Table 5: Examples of README similarity scores.

Repo 1	Repo 2	Score
tshamz/gulp-shopify-upload-with-callbacks	mikenorthorp/gulp-shopify-upload	0.91
kosmoport/generator-kosmoport	jackong/generator-webpack-library	0.98

in Figure 2. Table 5 shows examples of pairs of repositories we discovered to be similar this way. The corresponding variable in our models is `Num_activated_`$\mathcal{G}_{descsim}$`_neighbors`.

Competition. For every node in our network, we operationalize the presence of competing tools (H_6) as `Has_competitor_badge`, a binary variable capturing whether or not the repository has already adopted any competing tool from the same equivalence class, as classified in Table 1.

Global tool adoption. To measure the overall popularity of tools (H_4), we compute `Num_badges_adopted_globally` as the total number of badges that have been adopted by any repository in our dataset up until the current time window.

Control variables. In addition to the concepts directly involved in the hypotheses in Section 3, we operationalize as control variables the following constructs: (1) A repository's *proclivity to adopt a badge*—binary variable `Has_other_badges`, set as True if the repository has adopted any badge before. (2) A repository's overall level of *project activity*—numeric variable `Num_commits`; since badges are added to a repository through a commit, the more commits in a repository, the higher the chances to adopt a badge. (3) A repository's *popularity*—numeric variable `Num_stars`.

6 ANALYSIS

A variety of computational approaches exist for analyzing diffusion processes and how innovations spread. Some models are *macroscopic*, *e.g.*, SIR (susceptible-infected-recovered) and SIS (susceptible-infected-susceptible) [33] in epidemiology, and the Bass model [7]. For example, we fit a standard Bass diffusion model [7] to our longitudinal adoption data for each of the 12 quality assurance tools in our sample. That is, we model the total number of projects having adopted a badge until time t as a function of two parameters: $F(t) = (1 - e^{-(p+q)t})/(1 + \frac{q}{p}e^{-(p+q)t})$, where p is interpreted as the coefficient of "innovation," capturing the rate at which new adopters that have not been influenced by previous adopters, adopt the innovation; and q is interpreted as the "imitation" rate, capturing the rate at which new adopters adopt the innovation after coming in

contact with the previous adopters [75]. Fitting the model reveals generally low values and little difference between the estimates of p, *i.e.*, little "innovation," and generally high values with variance between the estimates of q, *i.e.*, high "imitation," or contagion (Figure 1). While such models can offer a macroscopic picture of the adoption of a tool over time, they are quite limited — no network, no project characteristics, *etc.*

There are also more powerful longitudinal *event history analysis* models that estimate the relative impact of multiple time-constant and time-varying factors on the likelihood of a person adopting a particular innovation at a given time [50, 67]. Moreover, such models can also be combined with social network analysis techniques to study how innovations spread through a network, *e.g.*, by modeling the network exposure as one of the effects. Two types of network exposure modeling approaches are common among social network analysis researchers: (1) *independent cascade models* [26] assume that each node has a fixed probability of adopting a particular innovation or behavior; and (2) *threshold models* [29] assume that a node in the network adopts a particular innovation only if the fraction of neighbors who have already adopted is greater than a certain threshold. Both models have generally been used in conjunction with viral marketing or influence maximization applications [42]; that is, given limited budget to promote a certain product, the goal is to find which people to "infect" first such that the overall adoption of the product in the network is maximized.

In our case, the network data is continuous, therefore we can model exposure continuously rather than as a threshold, *i.e.*, for any repository node at a given point in time, we can observe the number of neighboring repositories in a given graph (Section 5.3) that have previously adopted the tool. Based on these measures and the operationalizations in Section 5.4 above, we use *survival analysis*, a popular multivariate event history modeling technique, to estimate the average hazard rate of a repository adopting a particular tool at a given time and the relative strength of the effects of the different factors considered. We are not aware of other software engineering research combining social network analysis with survival analysis this way; however, in other domains, interested readers can refer to studies of the spread of obesity [14], happiness [23], new products [35], memes [27], or health-related interventions [83] through social networks for more examples of comparable computational approaches.

Survival analysis specifics. In our case an "event" is the adoption of a tool, for which we can observe the adoption time. However, if the event has not occured for a particular repository until the end of the observation period, we cannot infer anything about the chances of it occurring in the future. Survival analysis techniques are designed to handle such *right censored* data effectively [48].

To make the problem more tractable, rather than updating our variables after every new event, we compute feature values only in monthly (4-week) windows; *e.g.*, the number of commits variable for a given window is the cumulative number of commits in the repository by the end of this window. The observation period ends at the window in which the tool is adopted—for adopters—or at the time of the repository's last commit—for repositories yet to adopt.

We model jointly the effects of the different factors above on the time to adoption of a tool by a given repository, while controlling for

covariates. For each repository, we have a *survival time T* on record (number of months until the binary dependent variable *Adoption* becomes True). The probability of reaching a given survival time t is given by the *hazard rate* $h(t) = \frac{P(T < t + \delta t | T \geq t)}{\delta t}$. We use Cox proportional hazard models [48] to estimate this probability and the coefficients of the regression $h(t, X) = \theta(t) \exp(\beta^T X)$ using partial likelihood, without making any assumptions about the baseline hazard rate [37, 48].

As validation, we tested our models for multicollinearity and removed predictors if they had a variance inflation factor (VIF) above 5 [15]. Two variables consistently exceeded this threshold: the number of other repositories in the same organization as the focal repository, that have adopted the same tool before—a control for organization structure we considered including—was collinear with `Num_activated_`$\mathcal{G}_{CN}$`_neighbors`; and the overall popularity of a given tool itself—another control we considered including—was collinear with `Num_badges_adopted_globally`. In other cases, some variables exceeded this threshold only in some models; these collinear variables are the ones omitted from Tables 6 and 7.

Early adopters. We use the survival models described above to test hypotheses H_2–H_7. For the remaining H_1 (characteristics of early adopters), we estimate a logistic regression model, with a binary dependent variable `Is_early_adopter` and the variables described above in Section 5.4 as independent variables.

7 RESULTS

Overall, we built survival models for each of the 12 quality assurance tools and 5 control-group badges in Table 1. Given limited space, we only present the survival modeling summaries for six representative tools and two representative non-tool badges for comparison in Tables 6 and 7, but summarize and discuss the patterns we observe across all models. We also report separately on the logistic regression model for early adopters. Our replication package[6] contains all the modeling results discussed but not included here.

For every variable in the survival models in Tables 6 and 7, we show in the "HR (St Err)" column the Hazard Ratios, *i.e.*, the exponentiated estimated regression coefficients β, together with the standard errors and the statistical significance p-value ranges (***$p < 0.001$, **$p < 0.01$, *$p < 0.05$) for the coefficients. One can interpret these directly, *e.g.*, if $\beta_i = 0.3$ and statistically significant, then a unit increase in the variable X_i increases the adoption hazard on average by $HR = \exp(0.3)$, *i.e.*, by a factor of 1.35, *while holding all the other variables fixed.*[7] We also report in the "LR Chisq" column the amount of variance explained by each variable, as derived from an ANOVA analysis. One can interpret these as a measure of a variable's effect size, by computing the fraction of variance explained by that variable relative to the total amount of variance explained by the model, *i.e.*, the sum of "LR Chisq" values.

H_1: Early adopters (✓). We find strong evidence supporting our hypothesis that there are considerable differences in social standing and technological openness between early adopters and other GITHUB users. The total number of languages (4.5% increase in odds per language), stars (2.4% increase per 1,000), and followers

(10% per 1,000) are significantly and positively associated with the odds of being an early adopter. Among the control variables, older users are more likely to be early adopters (3.9% increase per month); the number of commits does not have a significant effect.

H_2: Social network ties (✓). For most tools, both social diffusion channels (the committer and watcher networks), after controlling for other factors, had a significant effect on whether the tool was adopted. The more projects that a particular repository is socially connected to, that already adopted the tool, the greater the probability it will also adopt the tool. As hypothesized, the committer network has the strongest effects. In all the models in Tables 6 and 7, the `Num_activated_`$\mathcal{G}_{CN}$`_neighbors` variable is always significant and has much larger effects than `Num_activated_`$\mathcal{G}_{WN}$`_neighbors`.

H_3: Technical network ties (✓). We found that if a repository includes as a dependency any other repository that has already adopted the tool, then there is a higher probability for the repository to adopt that tool as well, holding other variables fixed. For example, this can be clearly seen in Table 6 for COVERALLS: the adoption "hazard" increases by a factor 1.73 for every factor e increase in the number of direct dependencies (note the log-transformed `Num_activated_`$\mathcal{G}_{dep}$`_neighbors`).

H_4: Global tool adoption (✓/✗). For all tools but TRAVIS, the more widely adopted other tools and badges are globally, the more likely a repository is to also adopt the tool, in line with the hypothesis. The TRAVIS exception can perhaps be explained by the tool's absolutely highest popularity among all the tools we considered.

We also observe that if a repository has adopted a tool (badge) before, then it was highly probable that the repository will go on and adopt other tools (badges); note the hazard ratios above 1 for `Has_other_badges` across all models.

H_5: Functional similarity (✓). In our models, we discover that although functional similarity as captured by similar dependencies (`Num_activated_`$\mathcal{G}_{depsim}$`_neighbors`) does not have a strong effect on adoption for most tools, user-defined similarity as captured by similar READMEs (`Num_activated_`$\mathcal{G}_{descsim}$`_neighbors`) is highly significant in almost all adoption scenarios, with sizable positive effects (higher likelihood of adoption when similar repositories have already adopted). The exceptions are CODESHIP and CODECLIMATE, which may not have sufficient data to detect these effects.

H_6: Competition (✓/✗). Competition has significant effects in most applicable cases. In our models `Has_competitor_badge` is significant and typically negatively correlated with the outcome, *i.e.*, if a repository has adopted a competing tool, then the likelihood of the repository adopting the focal tool decreases significantly. Continuous integration tools are most illustrative, given the strong competition among TRAVIS, CIRCLE, and CODESHIP: Repositories are less likely to adopt any one of these tools if they have already adopted another of one of these tools. APPVEYOR is the exception, with no significant effect. We expect that since APPVEYOR is a Windows-specific tool, maintainers may adopt it in addition to other CI tools, not in place of, but there is insufficient evidence in our model to conclude either way. The APPVEYOR model also shows strong user-defined similarity effects (`Num_activated_`$\mathcal{G}_{descsim}$`_neighbors`), further suggesting Windows specificity.

---

[6]https://github.com/BadgeDiffuser/badge_diffusion_supplementary
[7]Note that some variables have been log-transformed to reduce heteroscedasticity, which should be accounted for in the interpretation.

Table 6: Survival models for the four competing continuous integration tools in Table 1.

	TRAVIS		CIRCLE		CODESHIP		APPVEYOR	
	response: Adoption = T		*response*: Adoption = T		*response*: Adoption = T		*response*: Adoption = T	
	HR (St Err)	LR Chisq	HR (St Err)	LR Chisq	HR (St Err)	LR Chisq	HR (St Err)	LR Chisq
N_commits (log)	$1.58 (0.00)^{***}$	8224.62^{***}	$1.44 (0.02)^{***}$	291.79^{***}	$1.49 (0.05)^{***}$	68.10^{***}	$1.37 (0.02)^{***}$	188.53^{***}
N_stars (log)	$0.89 (0.00)^{***}$	659.32^{***}	$0.92 (0.02)^{***}$	22.93^{***}	$0.96 (0.05)$	0.70	$0.93 (0.02)^{***}$	15.33^{***}
Has_other_badges	$1.97 (0.01)^{***}$	3094.72^{***}	$1.91 (0.05)^{***}$	158.65^{***}	$1.52 (0.13)^{**}$	10.15^{**}	$4.65 (0.08)^{***}$	532.32^{***}
N_badges_globally (log)	$0.92 (0.00)^{***}$	449.12^{***}	$2.18 (0.05)^{***}$	316.97^{***}	$1.11 (0.06)$	3.11	$1.29 (0.04)^{***}$	45.31^{***}
Has_competitor_badge	$0.01 (0.32)^{***}$	1528.14^{***}	$0.09 (0.12)^{***}$	815.57^{***}	$0.16 (0.25)^{***}$	84.07^{***}	$0.97 (0.05)$	0.30
N_activ_$\mathcal{G}_{WN}$_neighb (log)	$1.00 (0.00)$	0.57	$1.04 (0.03)$	1.63	$1.15 (0.12)$	1.23	$1.07 (0.02)^{**}$	8.43^{**}
N_activ_$\mathcal{G}_{CN}$_neighb (log)	$1.37 (0.00)^{***}$	4237.64^{***}	$2.14 (0.03)^{***}$	404.49^{***}	$2.63 (0.07)^{***}$	162.54^{***}	$1.92 (0.03)^{***}$	483.34^{***}
N_activ_$\mathcal{G}_{dep}$_neighb (log)	$1.18 (0.01)^{***}$	272.48^{***}	$1.65 (0.09)^{***}$	31.08^{***}	$1.27 (0.37)$	0.41	$1.95 (0.07)^{***}$	80.54^{***}
N_activ_$\mathcal{G}_{depsim}$_neighb (log)	$0.96 (0.02)^{*}$	4.14^{*}	$0.42 (0.39)^{*}$	5.91^{*}			$2.63 (0.24)^{***}$	12.20^{***}
N_activ_$\mathcal{G}_{descsim}$_neighb (log)	$1.46 (0.02)^{***}$	339.70^{***}	$2.06 (0.11)^{***}$	31.13^{***}			$5.41 (0.14)^{***}$	86.43^{***}

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

Table 7: Survival models for two competing code coverage tools and two control-group badges in Table 1.

	COVERALLS		CODECLIMATE		LICENSE		NPM-DOWNLOADS	
	response: Adoption = T		*response*: Adoption = T		*response*: Adoption = T		*response*: Adoption = T	
	HR (St Err)	LR Chisq	HR (St Err)	LR Chisq	HR (St Err)	LR Chisq	HR (St Err)	LR Chisq
N_commits (log)	$1.62 (0.01)^{***}$	2205.75^{***}	$1.70 (0.02)^{***}$	1002.49^{***}	$1.48 (0.02)^{***}$	497.82^{***}	$1.31 (0.01)^{***}$	641.90^{***}
N_stars (log)	$0.87 (0.01)^{***}$	249.54^{***}	$0.94 (0.01)^{***}$	19.26^{***}	$0.98 (0.02)$	2.50	$1.04 (0.01)^{***}$	24.67^{***}
Has_other_badges	$2.92 (0.03)^{***}$	1456.79^{***}	$2.70 (0.05)^{***}$	504.00^{***}	$2.14 (0.05)^{***}$	276.70^{***}	$2.53 (0.03)^{***}$	1002.85^{***}
N_badges_globally (log)	$1.12 (0.01)^{***}$	73.30^{***}	$1.07 (0.03)^{**}$	9.91^{**}	$2.39 (0.04)^{***}$	701.28^{***}	$1.67 (0.02)^{***}$	792.98^{***}
Has_competitor_badge	$1.13 (0.13)$	0.87	$2.36 (0.07)^{***}$	111.08^{***}				
N_activ_$\mathcal{G}_{WN}$_neighb (log)	$0.95 (0.01)^{***}$	28.45^{***}	$0.83 (0.02)^{***}$	92.56^{***}	$0.78 (0.02)^{***}$	121.53^{***}	$0.95 (0.01)^{***}$	28.41^{***}
N_activ_$\mathcal{G}_{CN}$_neighb (log)	$1.67 (0.01)^{***}$	1911.11^{***}	$1.86 (0.02)^{***}$	1051.27^{***}	$2.38 (0.02)^{***}$	1102.76^{***}	$1.78 (0.01)^{***}$	2683.36^{***}
N_activ_$\mathcal{G}_{dep}$_neighb (log)	$1.73 (0.03)^{***}$	351.03^{***}	$1.86 (0.05)^{***}$	132.14^{***}	$1.91 (0.07)^{***}$	77.80^{***}	$1.83 (0.03)^{***}$	385.91^{***}
N_activ_$\mathcal{G}_{depsim}$_neighb (log)	$0.89 (0.06)^{*}$	4.57^{*}	$1.06 (0.17)$	0.14	$1.36 (0.12)^{*}$	5.43^{*}	$0.96 (0.08)$	0.31
N_activ_$\mathcal{G}_{descsim}$_neighb (log)	$2.04 (0.05)^{***}$	138.70^{***}			$2.19 (0.06)^{***}$	110.80^{***}	$2.38 (0.04)^{***}$	295.42^{***}

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

Additionally, for coverage tools, we found that there is no competition effect. The only notable exception is CODECLIMATE, where the effect is reversed, *i.e.*, if a repository has adopted a competing tool, then it is highly likely to also adopt CODECLIMATE. One explanation could be that CODECLIMATE has more features than CODECOV and COVERALLS [41]. In addition, badges for CODECOV and COVERALLS look the same, while the badge for CODECLIMATE is visually distinct.

H$_7$: Tools vs badges (✗). Surprisingly, the two sets of hazard ratios corresponding to `Num_activated_`$\mathcal{G}_{CN}$`_neighbors` (one for the quality assurance tools we consider, and one for the control group tools/badges) appear qualitatively similar; see the models for LICENSE and NPM-DOWNLOADS in Table 7 and the rest in the online replication package. That is, there does not appear to be much difference between the factors affecting the diffusion of quality assurance tools and those affecting the spread of non-tool badges, contrary to our hypothesis.

8 DISCUSSION AND IMPLICATIONS

There is strong empirical evidence in the software engineering literature for the effectiveness of many tools and practices (*e.g.*, continuous integration [78], dependency management [49]). That is, in addition to individual companies that may want to promote their tools, it may be beneficial for an entire community or ecosystem to encourage the adoption of tools and practices, to improve productivity, sustainability, code quality, documentation quality, security, and many other qualities. Similarly, it may be in the best practice of a company to spread tools and best practices among all its teams [53].

Our results provide strong empirical support that social science theories of adoption of innovations, including Rogers' with many theorized relationships, also hold for software engineering tools used throughout the *npm* ecosystem. Concretely, we find supporting evidence for multiple factors that impact adoption: the increased social standing of early adopters (**H$_1$**); the legitimacy that comes from using popular tools (**H$_4$**); network ties, both social and technical (**H$_2$**, **H$_3$**); good context fit (**H$_5$**); and the absence of competitors (**H$_6$**). Practically, these results have potentially far-reaching *implications for open-source and industrial developers, as well as academic and commercial tool builders*, all of which could benefit from evidence-based insights into how to speed up tool adoptions.

One mechanism stands out as having particularly strong empirical evidence in our models of tool adoption across *npm* repositories: the network ties through which projects may pick up tooling-related cues from each other, especially the social dependencies via joint contributors. These networks, taken together with visible indicators of a tool's existence in an open-source project (*e.g.*,

badges), begets exposure to these tools: in the absence of competition, **the probability of a tool's adoption is proportional to the amount of exposure the tool has through social network ties**. This answers a question from the introduction ("How many developers must be exposed to a new tool or technology, and under what conditions, before it catches on and is adopted en masse?") — there is no threshold; the more developers get exposed, the higher the chances of adoption.

A practical recommendation that follows is that if you want to enable more adoption, one way is to **make the tool in question more visible**. Likewise, the converse seems to be true: if potentially meaningful tools and best practices aren't visible, they might not be uptaken and used, leading to a missed opportunity for adoption. There are many ways to increase visibility. For example, other means for tools to be made visible, beyond badges, include GitHub integrations and webhooks. There are also other latent sources of information and social influence beyond GitHub, *e.g.*, social media websites like Twitter, other online communities, the user's workplace, or friends. All of these factors can increase the visibility and exposure of developers to tools, and should be considered.

Another practical recommendation to increase adoption, beyond making the tool more visible to any particular potential user, is to **expose more potential users** to the tool in question. For example, to do this efficiently, stakeholders looking to promote tool adoption can consider strategies not far from viral marketing campaigns in advertising. This means targeting a select few nodes in the network, focusing on them for adoption of a tool or technique first, and due to natural diffusion, an increased rate of adoption may follow.

Another potentially actionable finding of this work is the apparent similarity in the diffusion processes between quality assurance tools and other non-tool badges (**H**$_7$). That is, analysis tools and CI seem to be adopted and uptaken as quickly as simple textual badges with no underlying automated validation or analysis. While we remain cautious of replication and confirmation on more data and on data from other ecosystems, the implication, if the result holds, we believe is quite significant for tool builders and policy makers: the relatively higher complexity of the quality assurance tools and relatively higher effort to adopt, which are largely intrinsic to the task and unavoidable, do not appear to be what's holding back adoption; instead, the factors driving adoption seem much more within one's control. Therefore, toolsmiths should be able to get complex tools as widely adopted as possible, if promoted appropriately; **tool complexity does not appear to be as much of a barrier to adoption as the theory predicts**. Practically, the recommendation is to invest in promotion and increased visibility.

Our results, while a promising first step, also have *implications for future academic research*. In particular, recalling Rogers's five attributes of innovations that influence their adoption, this work did not address *trialability*, which could also meaningfully affect diffusion. For example, cloud-based CI tools such as Travis or Circle could be considered more trialable than CI tools initially offered for standalone installation on a local server, such as Drone and Jenkins. In this case, the increased effort required to experiment with the non-cloud-based CI tools could hinder their spread.

Another potential future work direction is exploring the observability attribute of tools beyond badges. More research is needed to work out which factors contribute the most to increased visibility,

both among the ones afforded by the platform (*e.g.*, integrations and webhooks) as well as the latent ones (*e.g.*, social media). All of these are worthwhile directions.

Our study is limited to a single ecosystem, *npm*, which has a unique culture and community that is particularly open to innovations [9]. However, other software ecosystems can have vastly different cultures and practices [9]. Future studies could investigate differences in diffusion patterns in different software ecosystems in comparison to their unique cultures and communities.

Moreover, while we investigated whether quality assurance tools spread differently than non-tool badges, in reality even tools within the same task class may vary in terms of configuration overhead and features [41]; from the perspective of toolsmiths, it would worth studying which usability features allow more effective diffusion.

Finally, while we saw differences in social standing between adopter categories, *e.g.*, early and late adopters, this was also only a beginning. Future work may design, *e.g.*, case studies of particularly influential users who are effective in spreading new practices; however, "influential" users remain hard to define.

9 CONCLUSION

We performed a large-scale longitudinal analysis of diffusion of 12 quality assurance tools in the *npm* ecosystem. Grounded in diffusions of innovation theory, our results confirm the extent to which observability plays a key role in software tool adoption, as per the theory. We also provide an in-depth analysis using survival analysis-based regression models on the effect of multiple social-technical factors on the adoption of tools. We found that social factors such as exposure to repositories that have already accepted a tool contribute significantly the probability of tool adoption. We additionally provide a characterization to differentiate between early and late adopters. There we found characteristic differences between early adopters and late adopters–for example, early adopters tend to have more experience with more programming languages and programming paradigms, and they tend to have more stars and followers on GitHub. We also explored the effect of competition between tools and how that impacted diffusion. We found, in the context of CI tools, that once a repository chose a CI tool, it rarely adopted another one. Besides many concrete results, our study provides a robust and theoretically grounded framework to study the diffusion of other software engineering tools and practices through (social) networks.

SUPPLEMENTARY MATERIAL AND REPLICATION PACKAGE

A replication package for our analysis, also including the results discussed but not presented here, is available online at https://github.com/CMUSTRUDEL/badge_diffusion_supplementary.

REFERENCES

[1] Ritu Agarwal and Jayesh Prasad. 1997. The role of innovation characteristics and perceived voluntariness in the acceptance of information technologies. *Decision Sciences* 28, 3 (1997), 557–582.

[2] Ritu Agarwal and Jayesh Prasad. 2000. A field study of the adoption of software process innovations by information systems professionals. *IEEE Transactions on Engineering Management* 47, 3 (2000), 295–308.

[3] Sadika Amreen, Audris Mockus, Russell Zaretzki, Christopher Bogart, and Yuxia Zhang. 2020. ALFAA: Active Learning Fingerprint based Anti-Aliasing for correcting developer identity errors in version control systems. *Empirical Software Engineering* (2020), 1–32.

[4] James G Anderson and Stephen J Jay. 1985. The diffusion of medical technology: Social network analysis and policy research. *The Sociological Quarterly* 26, 1 (1985), 49–64.

[5] Sinan Aral, Lev Muchnik, and Arun Sundararajan. 2009. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proceedings of the National Academy of Sciences* 106, 51 (2009), 21544–21549.

[6] Emanoel Francisco Sposito Barreiros. 2016. *The epidemics of programming language adoption*. Ph.D. Dissertation. Universidade Federal de Pernambuco.

[7] Frank M Bass. 1969. A new product growth for model consumer durables. *Management Science* 15, 5 (1969), 215–227.

[8] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. 2010. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM* 53, 2 (2010), 66–75.

[9] 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.

[10] Kurt D Bollacker, Steve Lawrence, and C Lee Giles. 1998. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proc. International Conference on Autonomous Agents (AGENTS)*. 116–123.

[11] Ronald S Burt. 2004. Structural holes and good ideas. *Amer. J. Sociology* 110, 2 (2004), 349–399.

[12] Richard O Carlson. 1964. School superintendents and adoption of modern math: a social structure profile. *Innovation in education. New York: Teacher's College, Columbia University* (1964).

[13] Frank KY Chan and James YL Thong. 2009. Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems* 46, 4 (2009), 803–814.

[14] Nicholas A Christakis and James H Fowler. 2007. The spread of obesity in a large social network over 32 years. *New England Journal of Medicine* 357, 4 (2007), 370–379.

[15] Patricia Cohen, Stephen G West, and Leona S Aiken. 2014. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press.

[16] Laura Dabbish, Colleen Stuart, Jason Tsay, and James Herbsleb. 2012. Leveraging transparency. *IEEE Software* 30, 1 (2012), 37–43.

[17] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*. ACM, 1277–1286.

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

[19] Premkumar Devanbu, Thomas Zimmermann, and Christian Bird. 2016. Belief & evidence in empirical software engineering. In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, 108–119.

[20] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. 2020. Detecting and Characterizing Bots that Commit Code. In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM.

[21] Paul J DiMaggio and Walter W Powell. 1983. The iron cage revisited: Institutional isomorphism and collective rationality in organizational fields. *American Sociological Review* (1983), 147–160.

[22] Robert G Fichman and Chris F Kemerer. 1993. Adoption of software engineering process innovations: The case of object orientation. *Sloan Management Review* 34 (1993), 7–7.

[23] James H Fowler and Nicholas A Christakis. 2008. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study. *BMJ* 337 (2008), a2338.

[24] Jay Galbraith. 1973. Designing complex organizations. (1973).

[25] Keheliya Gallaba and Shane McIntosh. 2018. Use and misuse of continuous integration features: An empirical study of projects that (mis) use Travis CI. *IEEE Transactions on Software Engineering* (2018).

[26] Jacob Goldenberg, Barak Libai, and Eitan Muller. 2001. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters* 12, 3 (2001), 211–223.

[27] Manuel Gomez-Rodriguez, Jure Leskovec, and Bernhard Schölkopf. 2013. Modeling information propagation with survival theory. In *Proc. International Conference on Machine Learning*. 666–674.

[28] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: GitHub's data from a firehose. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, 12–21.

[29] Mark Granovetter. 1978. Threshold models of collective behavior. *Amer. J. Sociology* 83, 6 (1978), 1420–1443.

[30] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A survey of software learnability: metrics, methodologies and guidelines. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 649–658.

[31] Morten T Hansen. 1999. The search-transfer problem: The role of weak ties in sharing knowledge across organization subunits. *Administrative Science Quarterly* 44, 1 (1999), 82–111.

[32] 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.

[33] Herbert W Hethcote. 2000. The mathematics of infectious diseases. *SIAM Rev.* 42, 4 (2000), 599–653.

[34] Susan H Higgins and Patrick T Hogan. 1999. Internal diffusion of high technology industrial innovations: an empirical study. *Journal of Business & Industrial Marketing* 14, 1 (1999), 61–75.

[35] Raghuram Iyengar, Christophe Van den Bulte, and Thomas W Valente. 2011. Opinion leadership and social contagion in new product diffusion. *Marketing Science* 30, 2 (2011), 195–212.

[36] Ciera Jaspan, I-Chin Chen, and Anoop Sharma. 2007. Understanding the value of program analysis tools. In *Companion Proc. ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications*. 963–970.

[37] Stephen P Jenkins. 2005. Survival analysis. *Unpublished manuscript, Institute for Social and Economic Research, University of Essex, Colchester, UK* 42 (2005), 54–56.

[38] 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 *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 73–84.

[39] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, 672–681.

[40] Richard Johnson. 1999. Applying the technology acceptance model to a systems development methodology. *Proc. AMCIS* (1999), 197.

[41] David Kavaler, Asher Trockman, Bogdan Vasilescu, and Vladimir Filkov. 2019. Tool choice matters: JavaScript quality assurance tools and usage outcomes in GitHub projects. In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, 476–487.

[42] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 137–146.

[43] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark GJ Van Den Brand. 2012. Who's who in Gnome: Using LSA to merge software repository identities. In *Proc. International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 592–595.

[44] Lauren Lanahan and Daniel Armanios. 2018. Does more certification always benefit a venture? *Organization Science* 29, 5 (2018), 931–947.

[45] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. 2019. World of code: an infrastructure for mining the universe of open source VCS data. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, 143–154.

[46] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. 2011. Mind your language: on novices' interactions with error messages. In *Proc. Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 3–18.

[47] Leo A Meyerovich and Ariel S Rabkin. 2013. Empirical analysis of programming language adoption. In *Proc. International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*. ACM, 1–18.

[48] Rupert G Miller Jr. 2011. *Survival analysis*. Vol. 66. John Wiley & Sons.

[49] Samim Mirhosseini and Chris Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *Proc. International Conference on Automated Software Engineering (ASE)*. IEEE, 84–94.

[50] Mark R Montgomery and John B Casterline. 1998. Social networks and the diffusion of fertility control. (1998).

[51] Emerson Murphy-Hill and Andrew P Black. 2008. Breaking the barriers to successful refactoring: observations and tools for extract method. In *Proc. International Conference on Software Engineering (ICSE)*. 421–430.

[52] Emerson Murphy-Hill and Gail C Murphy. 2011. Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In *Proc. ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*. ACM, 405–414.

[53] Emerson Murphy-Hill, Edward K Smith, Caitlin Sadowski, Ciera Jaspan, Collin Winter, Matthew Jorde, Andrea Knight, Andrew Trenk, and Steve Gross. 2019. Do developers discover new tools on the toilet?. In *Proc. International Conference*

on Software Engineering (ICSE). IEEE, 465–475.

[54] Ramana Nanda and Jesper B Sørensen. 2010. Workplace peers and entrepreneurship. *Management Science* 56, 7 (2010), 1116–1126.

[55] William Ocasio. 1997. Towards an attention-based view of the firm. *Strategic Management Journal* 18, S1 (1997), 187–206.

[56] Jeffrey Pfeffer. 1976. Beyond management and the worker: The institutional function of management. *Academy of Management Review* 1, 2 (1976), 36–46.

[57] Jeffrey Pfeffer and R Gerald. 1978. Salancik. 1978. The external control of organizations: A resource dependence perspective.

[58] Akond Rahman, Asif Partho, David Meder, and Laurie Williams. 2017. Which factors influence practitioners' usage of build automation tools?. In *Proc. International Workshop on Rapid Continuous Software Engineering (RCoSE)*. IEEE, 20–26.

[59] Colin Renfrew. 1989. The origins of Indo-European languages. *Scientific American* 261, 4 (1989), 106–115.

[60] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How do professional developers comprehend software?. In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, 255–265.

[61] Everett M Rogers. 2010. *Diffusion of innovations*. Simon and Schuster.

[62] Everett M Rogers and D Lawrence Kincaid. 1981. Communication networks: Toward a new paradigm for research. (1981).

[63] Bryce Ryan and Neal C Gross. 1943. The diffusion of hybrid seed corn in two Iowa communities. *Rural Sociology* 8, 1 (1943), 15.

[64] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Söderberg, and Collin Winter. 2015. Tricorder: Building a program analysis ecosystem. In *Proc. International Conference on Software Engineering (ICSE)*, Vol. 1. IEEE, 598–608.

[65] Mali Senapathi. 2010. Adoption of software engineering process innovations: The case of agile software development methodologies. In *International Conference on Agile Software Development*. Springer, 226–231.

[66] David Strang and John W Meyer. 1993. Institutional conditions for diffusion. *Theory and Society* 22, 4 (1993), 487–511.

[67] David Strang and Nancy Brandon Tuma. 1993. Spatial and temporal heterogeneity in diffusion. *Amer. J. Sociology* 99, 3 (1993), 614–639.

[68] Evan T Straub. 2009. Understanding technology adoption: Theory and future directions for informal learning. *Review of Educational Research* 79, 2 (2009), 625–649.

[69] Mark C Suchman. 1995. Managing legitimacy: Strategic and institutional approaches. *Academy of Management Review* 20, 3 (1995), 571–610.

[70] Bart Theeten, Frederik Vandeputte, and Tom Van Cutsem. 2019. Import2vec learning embeddings for software libraries. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, 18–28.

[71] Kristín Fjóla Tómasdóttir, Maurício Aniche, and Arie van Deursen. 2017. Why and how JavaScript developers use linters. In *Proc. International Conference on Automated Software Engineering (ASE)*. IEEE, 578–589.

[72] Omer Tripp, Salvatore Guarnieri, Marco Pistoia, and Aleksandr Aravkin. 2014. Aletheia: Improving the usability of static security analysis. In *Proc. ACM SIGSAC Conference on Computer and Communications Security*. 762–774.

[73] 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)*. ACM, 511–522.

[74] Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Brian P Bailey, and Ralph E Johnson. 2012. Use, disuse, and misuse of automated refactorings. In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, 233–243.

[75] Thomas W Valente. 1996. Network models of the diffusion of innovations. *Computational & Mathematical Organization Theory* 2, 2 (1996), 163–164.

[76] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. 2015. A Data Set for Social Diversity Studies of GitHub Teams. In *Proc. International Conference on Mining Software Repositories (MSR), Data Track*. IEEE, 514–517.

[77] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. 2014. On the variation and specialisation of workload? A case study of the Gnome ecosystem community. *Empirical Software Engineering* 19, 4 (2014), 955–1008.

[78] 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)*. 805–816.

[79] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, and Harald C Gall. 2018. Context is king: The developer perspective on the usage of static analysis tools. In *Proc. International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 38–49.

[80] David Widder, Michael Hilton, Christian Kästner, and Bogdan Vasilescu. 2019. A Conceptual Replication of Continuous Integration Pain Points in the Context of Travis CI. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM.

[81] Igor Scaliante Wiese, José Teodoro da Silva, Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2016. Who is who in the mailing list? Comparing six disambiguation heuristics to identify multiple addresses of a participant. In *Proc. International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 345–355.

[82] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying developers' adoption of security tools. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 260–271.

[83] Jiacheng Wu, Forrest W Crawford, David A Kim, Derek Stafford, and Nicholas A Christakis. 2018. Exposure, hazard, and survival analysis of diffusion on social networks. *Statistics in Medicine* 37, 17 (2018), 2561–2585.

[84] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social influences on secure development tool adoption: why security tools spread. In *Proc. ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*. ACM, 1095–1106.

[85] Ahmed Zerouali, Tom Mens, Jesus Gonzalez-Barahona, Alexandre Decan, Eleni Constantinou, and Gregorio Robles. 2019. A formal framework for measuring technical lag in component repositories—and its application to npm. *Journal of Software: Evolution and Process* 31, 8 (2019), e2157.

[86] Yang Zhang, Bogdan Vasilescu, Huaimin Wang, and Vladimir Filkov. 2018. One size does not fit all: an empirical study of containerized continuous deployment workflows. In *Proc. Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 295–306.

[87] Sedigheh Zolaktaf and Gail C Murphy. 2015. What to learn next: Recommending commands in a feature-rich environment. In *Proc. International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 1038–1044.