## Heard It through the GitVine: An Empirical Study of Tool Diffusion across the npm Ecosystem

<table>
<thead>
<tr>
<th>Hemank Lamba</th>
<th>Asher Trockman</th>
<th>Daniel Armanios</th>
<th>Christian Kästner</th>
</tr>
</thead>
<tbody>
<tr>
<td>University</td>
<td>University</td>
<td>University</td>
<td>University</td>
</tr>
<tr>
<td>Pittsburgh, PA, USA</td>
<td>Pittsburgh, PA, USA</td>
<td>Pittsburgh, PA, USA</td>
<td>Pittsburgh, PA, USA</td>
</tr>
</tbody>
</table>

### ABSTRACT

Automation 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

### ACM Reference Format:

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ESEC/FSE ’20, November 8–13, 2020, Virtual Event, USA © 2020 Copyright held by the owner/author(s).
https://doi.org/10.1145/3368089.3409705

---

### 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 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 BitBucket 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 (X) 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 quantitatively answering 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 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.

<table>
<thead>
<tr>
<th>Table 1: Overview of the software quality assurance tools in our study, organized by task equivalence class; see Sec. 5.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Cont. integration</td>
</tr>
<tr>
<td>Dependency ingmt</td>
</tr>
<tr>
<td>Code coverage</td>
</tr>
<tr>
<td>Cross browser test</td>
</tr>
</tbody>
</table>

**Control group** Slack, Gitter, License, npm-version, npm-downloads

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].
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 be adopted.

Rogers’ 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, 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:

**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:

**H1.** 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:

**H2.** 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:

**H3.** 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:

**H4.** 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:

**H5.** 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:

**H6.** 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 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: 

H2. 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 GrHub (over half a million at the time of data collection) by tracking the spread of the corresponding repository badges, e.g., build passing, coverage 98%, and dependencies up to date, using the badges implies using the underlying tools.2

We chose GrHub 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 project on a platform like GrHub 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 H1 (recall, this is the only developer-level hypothesis in our study; the others are project-level). For hypotheses H2–H6, we analyze the network’s adoption history using survival analysis, to model and 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 H7 we qualitatively4 compare the coefficient estimates for the social-diffusion-related variables in our models between the 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” (Table 1) of tools with relatively much simpler setup/usage and no underlying analyses, 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 for the experimental group appears higher than for the control group, we would consider H7 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 GrHub API, and git logs; it contains all npm packages which could be cross-linked to their corresponding GrHub 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 H7, 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.

---

2The 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.

3For example, https://github.com/BraveUX/for-the-badge.

4The relatively small size of our two samples, 12 experimental tools and 5 control group tools/badges, precludes sound statistical comparison.
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 down-stream 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 GrHub 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.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Num. nodes</th>
<th>Num. edges</th>
<th>Avg. degree</th>
<th>Avg. cluster coeff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Committer (GC)</td>
<td>148,582</td>
<td>4,461,796</td>
<td>30.03</td>
<td>0.80</td>
</tr>
<tr>
<td>Watcher (GW)</td>
<td>150,857</td>
<td>16,001,804</td>
<td>106.01</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Committer graph (social). Developers committing to multiple repositories can create strong social connections between the repositories, through which practices can spread (recall H5 above). For our committer graph (GC), 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 R1 at time t1 and then to R2 at time t5, resulting in an edge R1 → R2, available at t5 and later. We then compute, and model in our regressions below, the variable Num activated GC_neighbors, counting the number of a node’s neighbors in GC 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 (H6). We add an edge (in the watcher graph GW) 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 R2 at time t3, and then starts committing to R3 at time t6, thus creating the edge R2 → R3 starting t6. The corresponding variable in our models, computed analogously, is Num activated GW_neighbors.

Note how the two social graphs, GC and GW, capture different ends of the social connection spectrum: GC is expected to indicate the strongest connection between repositories, and GW 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 (H3), i.e., npm packages reusing other npm packages as libraries, we also extract a repository–repository dependency graph Gdep from the corresponding package JSON files. For example, in Figure 2, R2 is directly dependent on both R3 and R4. The corresponding variable in our models is Num activated Gdep_neighbors.

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 H1—recall the difference in study design for testing this hypothesis, discussed in Section 4—followed by operationalizations related to repositories, corresponding to H4–H6.

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 (H1). 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 language 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% 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 (H3), 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 Gdep graph, e.g., R2 ↔ R3 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 Gdep_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 (H3), 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., R4 ↔ R5

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.

<table>
<thead>
<tr>
<th>Repo 1</th>
<th>Repo 2</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>blueapp/rotate-log</td>
<td>treelite/asuka:</td>
<td>0.95</td>
</tr>
<tr>
<td>winstonjs/winston</td>
<td></td>
<td></td>
</tr>
<tr>
<td>substack/node-mkdirp</td>
<td>winstonjs/winston</td>
<td></td>
</tr>
<tr>
<td>daily-rotate-file</td>
<td>substack/mkdirp,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>winstonjs/winston-daily-rotate-file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tj/commander.js</td>
<td></td>
</tr>
<tr>
<td>tshamz/gulp-shopify-upload-with-callbacks</td>
<td>shama/grunt-hub:chalk/chalk,</td>
<td>1.0</td>
</tr>
<tr>
<td>kosmoport/generator-kosmoport</td>
<td>jackong/generator-webpack-library</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Examples of README similarity scores.

<table>
<thead>
<tr>
<th>Repo 1</th>
<th>Repo 2</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>tshamz/gulp-shopify-upload-with-callbacks</td>
<td>mikenorthern/gulp-shopify-upload</td>
<td>0.91</td>
</tr>
<tr>
<td>kosmoport/generator-kosmoport</td>
<td>jackong/generator-webpack-library</td>
<td>0.98</td>
</tr>
</tbody>
</table>

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 Gdescim_neighbors.

**Control variables.** In addition to the concepts directly involved in the hypotheses in Section 3, we operationalized 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 previously adopted any competing tool from the same equivalence class, as classified in Table 1.

**Global tool adoption.** To measure the overall popularity of tools (Hl), 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.

**Survival analysis specifics.** 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.
covariates. For each repository, we have a survival time $T$ on record (number of months until the binary dependent variable $\text{Adoption}$ becomes True). The probability of reaching a given survival time $t$ is given by the hazard rate $h(t) = \frac{f(t)}{S(t)}$. 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 $\text{Num\_activated\_GC\_N\_neighbors}$; and the overall popularity of a given tool itself—another control we considered including—was collinear with $\text{Num\_badges\_adopted\_globally}$. 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 $H_1$—$H_7$. For the remaining $H_1$ (characteristics of early adopters), we estimate a logistic regression model, with a binary dependent variable $\text{Is\_early\_adopter}$ and the variables described above in Section 5.4 as independent variables.

## 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 $\beta$, together with the standard errors and the statistical significance $p$-value ranges ($** p < 0.001$, "*" $p < 0.01$, "$^\circ$" $p < 0.05$) for the coefficients. One can interpret these directly, e.g., if $\beta_1 = 0.3$ and statistically significant, then a unit increase in the variable $X_1$ 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 ($\checkmark$). We find strong evidence supporting our hypothesis that there are considerable differences in social standing and technological openness between early adopters and other GrtHub 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 ($\checkmark$). 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 have 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 $\text{Num\_activated\_GC\_N\_neighbors}$ variable is always significant and has much larger effects than $\text{Num\_activated\_GWN\_neighbors}$.

$H_3$: Technical network ties ($\checkmark$). 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 Codeclimate: the adoption “hazard” increases by a factor 1.73 for every factor $e$ increase in the number of direct dependencies (note the log-transformed $\text{Num\_activated\_Gdep\_neighbors}$).

$H_4$: Global tool adoption ($\checkmark / X$). 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 $\text{Has\_other\_badges}$ across all models.

$H_5$: Functional similarity ($\checkmark$). In our models, we discover that although functional similarity as captured by similar dependencies ($\text{Num\_activated\_Gdepsim\_neighbors}$) does not have a strong effect on adoption for most tools, user-defined similarity as captured by similar READMEs ($\text{Num\_activated\_Gdescsim\_neighbors}$) 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 ($\checkmark / X$). Competition has significant effects in most applicable cases. In our models $\text{Has\_competitor\_badge}$ 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. Approveyor is the exception, with no significant effect. We expect that since Approveyor 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 Approveyor model also shows strong user-defined similarity effects ($\text{Num\_activated\_Gdescsim\_neighbors}$), 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.
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.

H₁: Tools vs badges (X). Surprisingly, the two sets of hazard ratios corresponding to Num activated GCN 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 (H₁); the legitimacy that comes from using popular tools (H₄); network ties, both social and technical (H₂, H₃); good context fit (H₃); and the absence of competitors (H₅). 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 networks. 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 (H7). 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—e.g., 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.

ACKNOWLEDGEMENTS
Lamba and Vasilescu have been supported in part by the NSF (awards 1717415, 1901311). Kästner has been supported in part by the NSF (awards 1318808, 1552944, and 1717022).

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/CMISTRUDEL/badge_diffusion_supplementary.
REFERENCES


