



Adding Sparkle to Social Coding: An Empirical Study of Repository Badges in the *npm* Ecosystem

Asher Trockman,[†] Shurui Zhou,[‡] Christian Kästner,[‡] Bogdan Vasilescu[‡]

[†]University of Evansville, USA

[‡]Carnegie Mellon University, USA

ABSTRACT

In fast-paced, reuse-heavy, and distributed software development, the transparency provided by social coding platforms like GITHUB is essential to decision making. Developers infer the quality of projects using visible cues, known as *signals*, collected from personal profile and repository pages. We report on a large-scale, mixed-methods empirical study of *npm* packages that explores the emerging phenomenon of repository badges, with which maintainers signal underlying qualities about their projects to contributors and users. We investigate which qualities maintainers intend to signal and how well badges correlate with those qualities. After surveying developers, mining 294,941 repositories, and applying statistical modeling and time-series analyses, we find that non-trivial badges, which display the build status, test coverage, and up-to-dateness of dependencies, are mostly reliable signals, correlating with more tests, better pull requests, and fresher dependencies. Displaying such badges correlates with best practices, but the effects do not always persist. In short,  .


1 INTRODUCTION

Contemporary software development is characterized by increased reuse and speed. Open-source software forges like GITHUB host millions of repositories of libraries and tools, which developers reuse liberally [25], creating complex, often fragile networks of interdependencies [11]. This has earned GITHUB a reputation as a one-stop shop for software development [39] and as an influencer of practices in both open-source and industry [33]. The DevOps culture [31, 46] also contributes to this acceleration, with its emphasis on automation and rapid deployment. As a result, developers are expected to make more decisions at higher speed, e.g., finding which libraries to depend on and which projects to contribute to.

A key enabler of this decision making process is the *transparency* provided by social coding platforms like GITHUB [20, 21]. The development history of open-source GITHUB projects is archived and publicly accessible in a standardized format, and user pages display aggregate information about one’s contributions and social standing in the community (e.g., through *stars* and *watchers*). This

transparency can enhance collaboration and coordination [21]. Using visible cues—known in the literature as *signals*—collected from personal profile and repository pages, developers can better manage their projects and dependencies, communicate more efficiently, become informed about action items requiring their attention, learn, socialize, and form impressions about each other’s coding ability, personal characteristics, and interpersonal skills [21, 38, 40, 57].

However, open-source ecosystems are also competitive. In order to survive and thrive, projects must successfully attract and retain contributors, and fend off competitors [16, 36, 42, 45]. In a social coding environment, the visible signals enabled by transparency can, therefore, be seen as a survival mechanism, with high profile signalers benefiting the most. For example, more popular and famous projects attract more contributors [62], coding “rock stars” collect thousands of followers [20], and visible traces of developer actions and interactions are used in recruitment and hiring [13, 37].

Here we focus on *repository badges*, images such as , embedded into a project’s README, often generated on-demand, reflecting the current status of online services the project is using, e.g., continuous integration and dependency management. From a *signaling theory* [52] perspective (§2), badges can be seen as easily observable signals used by maintainers to convey underlying qualities of their projects, e.g., code quality and adherence to best practices. The resulting increased transparency (hard to observe qualities become salient) may impact users’ and contributors’ decision making and the project’s chances of survival. Badges can also be seen as a *gamification mechanism* [23], i.e., a game-like incentive designed to engage participants (§2); e.g., a badge with real-time code coverage information may act as an incentive for contributors to improve the project’s test suite. In summary, badges are a potentially impactful feature in transparent, social coding environments. However, the value and effects of badges are not well understood.

In this paper, we explore two main research questions regarding badges. First, we explore the phenomenon quantitatively and qualitatively, and ask (RQ₁) *What are the most common badges and what does displaying them intend to signal?* Second, we analyze whether badges indeed signal what developers expect, and ask (RQ₂) *To what degree do badges correlate with qualities that developers expect?* To this end, we perform a large-scale mixed-methods empirical study of the badges in *npm*, a large and vibrant open-source ecosystem for JavaScript with documented interdependency-related coordination challenges [11], wherein many badges originated. We observe the frequency and historical adoption of badges among 294,941 *npm* packages, we survey maintainers and contributors about their intentions and perceptions, and we build regression models to test hypotheses regarding developer perceptions (collected when exploring RQ₁), such as, “coverage badges signal the importance of tests and therefore attract more pull requests with tests.”

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

ICSE ’18, May 27–June 3, 2018, Gothenburg, Sweden

© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.

ACM ISBN 978-1-4503-5638-1/18/05...\$15.00

<https://doi.org/10.1145/3180155.3180209>

Our investigation reveals that badges are popular in *npm*, adopted in 46 % of packages. The most frequent show the build status or version of the latest release, but dependency managers, code coverage, and download counts are also common. Maintainers intend to signal various qualities with badges, and, indeed, we found among others that build-status and coverage badges correlate with larger test suites and encourage external contributors to include more tests, popularity badges correlate with future gains in downloads, and the introduction of dependency-manager badges correlates with a lasting improvement in dependency freshness. Correlations are particularly strong for *assessment signals*, i.e., badges that test an underlying quality rather than just stating intentions. Our results provide guidance for package maintainers to make more informed decisions about badge adoptions, being more deliberate about what they intend to signal and how that signal is supported. For users and contributors, our results indicate which badges provide reliable signals.

In summary, we contribute (1) a survey among *npm* developers, (2) a large scale analysis of 294,941 *npm* packages and their history, and (3) an in-depth analysis (using multiple regression models and time-series regression discontinuity designs) of 8 hypotheses regarding the effect of badges on various qualities, showing many badges are indeed reasonably reliable signals. Furthermore, (4) we frame our discussion in the context of signaling theory and confirm that badges based on assessment signals are more reliable.

2 THEORETICAL FRAMEWORK

We argue that badge are intended to signal an underlying quality about the project to potential users and contributors. In addition, certain badges (e.g., code coverage) may encourage certain kinds of practices, or attempts to improve visible scores. Therefore, we frame our study in the context of signaling theory and gamification.

Signaling theory. Signaling theory is widely applied to *selection scenarios* in a range of disciplines, from economics [51] to biology [63]. In these scenarios, the *signaler* benefits from actions taken by the *receiver* (e.g., being selected over some alternative), which would not have occurred in the absence of the *signal*. Signals are observable pieces of information that indicate a hidden quality of the signaler. Receiver cost to interpret the signal tends to outweigh signal accuracy, with less reliable but more easily obtainable signals being preferred by receivers over more reliable signals that are costlier to observe or assess [28]. The classical example in economics is job market candidates signaling their quality through education: holding a degree from a prestigious institution is easily observable and communicates to potential employers the candidate’s ability, which is otherwise less readily observable [51]. Selection scenarios occur routinely in open-source, e.g., choosing which libraries to depend on [11], repositories to watch [49], developers to follow [10, 35] or hire [13, 37], and projects to contribute to [9, 14].

The phenomenon underlying signaling theory is *information asymmetry* [52], which occurs between those having all the information and those who could potentially make better decisions if they had all the information. To reduce information asymmetry, actors rely on observable signals (e.g., information in CVs). Information asymmetry also occurs in the open-source selection scenarios above. Even if activity traces are typically publicly accessible, not all information is equally accessible, with some requiring specialized mining, e.g., of git histories and issue trackers. For example,

to avoid outdated, possibly vulnerable dependencies, developers may adopt a dependency manager, e.g., GEMNASTIUM, DAVID [11, 41], to receive notifications when a dependency is updated; however, to potential users of the package, this practice is very difficult to recognize unless it is made obvious. A badge reporting the result of the same analysis, e.g., DAVID’s `dependencies up to date`, indicates publicly that the tool is not only enabled, but also used regularly. That is, we argue that *repository badges are signals*: by making certain information about the project’s code base or practices transparent, badges contribute to reducing information asymmetry between maintainers (insiders) and users and contributors (outsiders).

Research has confirmed that in transparent, social coding environments, observable signals in online profiles are used as indications of expertise and commitment [20, 38, 48, 55, 56], e.g., STACK OVERFLOW reputation score, GITHUB followers and longest activity streak. We expect that repository badges may have a similar effect.

Assessment signals. The literature distinguishes between *conventional signals* and *assessment signals* [24]. Both are used, but the former are not inherently reliable; the quality they indicate is established by convention and the signal is typically cheap to produce, therefore easy to fake. The latter are considered more reliable, because “the quality they signal is ‘wasted’ in the production of the signal, and the signal tends to be more expensive to produce for an individual with less of the quality” [48]. Therefore, for a signal to be effective, it must be both: (i) *observable*, i.e., readily noticeable by outsiders (otherwise it might go unnoticed), and (ii) *costly to produce*, such that only higher quality signalers can absorb the cost to produce it [17]. There is a great diversity of badges which, despite being equally observable on READMEs, vary widely in production cost. Some, e.g., DAVID’s `dependencies up to date`, indicate relatively deep technical qualities that are achieved with specific, arguably costly, practices. Others indicate technical or non-technical but relatively shallow qualities that are easy to look up elsewhere, e.g., `license BSD`, `npm v1.1.0`, and `Star 4k`, and others still are mere statements of intentions without any automated validation and thus without any associated costs, e.g., `code style standard` and `PRs welcome`.

Our goal is to evaluate the reliability of badges as signals, by assessing *signal fit* [17], i.e., the extent to which signals correspond to the desirable unobservable quality of the signaler, described by the strength of the statistical association between public information—the signal—and private information—the unobservable quality.

Gamification. Using game-design elements in non-gaming contexts [23] is mainstream in “social programming” [6, 54] environments, e.g., on Q&A sites like STACK OVERFLOW. These gamification elements are known to motivate existing users [3, 15, 27] as well as attract new users, at the detriment of other platforms [53, 58].

Despite being voluntarily displayed on project READMEs by maintainers, and not “earned” based on performance and automatically displayed by the platform, we argue that repository badges are also gamification mechanisms. Since badges typically represent best practices and tools (e.g., continuous integration, code coverage, dependency freshness), there is little incentive for maintainers to display badges indicative of “bad” practices (e.g., `build failing`, `coverage 1%`). Therefore, we expect that the mere presence of GITHUB badges correlates with best practices (i.e., they are in a sense “earned”). Furthermore, as we will discuss, badges such as `coverage 94%` may

incentivize contributors to follow better testing practices. Hence, GITHUB badges may have a similar effect to STACK OVERFLOW badges, steering user behavior towards specific practices [4].

Understanding practices in software ecosystems. Many studies have looked into specific practices in software ecosystems, including communication [8, 29, 32, 50], change planning [11, 12, 22, 44], dependency updates [5, 19, 30, 34, 41], static analysis [7, 64], testing and continuous integration [31, 60, 65], and many others. Badges and their underlying tools are also ecosystem-level practices. In contrast to prior work, however, we specifically take a broader view on badges as *signals* beyond individual tools and practices.

3 RQ₁: BADGES ON *npm*

We study the adoption and effects of badges in the *npm* ecosystem. *npm* is a package manager and corresponding repository launched in 2010, currently hosting package releases for 500,000+ distinct packages. It was originally designed for *Node.js* developers, but is used more broadly by many web developers today. As in other package managers and repositories, e.g., *Maven* and *RubyGems*, an *npm* package bundles files (typically JavaScript) with a README and metadata (*package.json*), which includes the unique release version and dependencies to other packages. A client installs and updates dependencies from the central *npm* repository. The *npm* community is generally considered innovation friendly, frequently adopts external packages, values making it easy to contribute and publish packages, and shows a healthy competition between multiple equivalent packages to solve any single problem [1, 11, 22, 61].

We chose *npm* because: (1) it provides API access to all package releases and metadata, including download counts, (2) most *npm* packages point to a GITHUB repository, (3) the *npm* registry and GITHUB both prominently show the package’s README file, providing a common place where badges are displayed, and (4) the *npm* community is innovation friendly and broadly experiments with and adopts developer services [11, 41], including cloud-based continuous integration, dependency managers, and badges.

3.1 Research Methods

To explore which badges are common and what they intend to signal (RQ₁), we conducted a survey and mined repositories at scale.

Survey design. To gauge perceptions and anticipated effects we designed two online surveys targeting *npm* package maintainers and corresponding GITHUB contributors; the former focused on what maintainers intend to signal about their packages by displaying badges and what effects, if any, they expect badges would have on their users and contributors, while the latter focused on what inferences contributors make about a package given its badges (for the specific questions see appendix Sec. ??). Both groups were asked to name specific badges when answering. We used mostly open-ended questions with free-text answers, and we piloted the survey first.

We extracted contact information and commit counts per person from the git logs of the 10,000 most popular *npm* packages by downloads, and classified developers as maintainers ($\geq 33\%$ of project commits) or contributors ($< 10\%$). We resolved multiple aliases using standard heuristics about common first name/last name/email formats [59]. We then randomly sampled two mutually exclusive

sets of contributors and maintainers, 300 each, and sent personalized invitation emails (580 succeeded, 294 to maintainers and 286 to contributors). We received 32 maintainer and 57 contributor responses, for a total response rate of 15.3%. Our respondents have a median 5 years of experience with open source and many surveyed maintainers have contributed to dozens of packages. We analyzed the textual responses using standard open-coding techniques.

Repository mining. We collected a multidimensional longitudinal data set of 294,941 *npm* packages, as follows. We started mining all *npm* packages on July 11, 2017 (512,834), then kept only 346,369 that: (1) had metadata on downloads, releases, dependencies, dependents, and maintainers and (2) linked to a GITHUB repository. In 11,316 cases when multiple packages linked to the same GITHUB repository,¹ we kept only the most downloaded one, which further reduced the size of our sample to 322,734. Next, we attempted to clone all GITHUB repositories locally, and succeeded for 294,941 packages; the others’ repositories were either private or missing.

To identify badges and their evolution, we used the git history of each repository’s README file.² This was iterative: We began by matching the markdown expression typically used to insert an SVG badge, i.e., `[![Badge Name](Image URL)](Service URL)`, but discovered packages with badges added as plain HTML, markdown *reference links*, and PNG. Consequently, we converted the markdown to HTML and matched `img` tags. To reduce false positives (not all images are badges), we curated a list of services frequently associated with badges, e.g., TRAVIS and COVERALLS, and devised regular expressions for classification. We then split all images into *other-badge* or *other-images* and iteratively refined the classification rules to define badge classes for specific services, such as TRAVIS. To support the iterative process, we generated web pages showing all found badges per class to inspect them manually for accuracy. As needed, we refined our classification until we reached stability (very few false positives in each class) and until the *other-badge* category comprised only obscure badges and non-badge images. Overall, we identified 88 kinds of badges (examples in Table 1). By analyzing READMEs longitudinally (following only first parents in the commit history so as not to detect temporary discrepancies between the mainline and other branches), we could identify the dates when badges were introduced or removed. Note that we analyzed badges on GITHUB, not *npm*. GITHUB provides a finer temporal granularity of commits, whereas *npm* typically shows the same README files but only updates them with each new release.

Threats to validity. As typical for a survey, our sample may suffer from selection bias. One should be careful when generalizing the results beyond the studied corpus of *npm* packages with corresponding GITHUB accounts. The *npm* community has certain characteristics and results may differ in other communities; e.g., Python and Java developers may adopt innovations at a different pace and may use different kinds of tools; not all package repositories show READMEs with badges as prominently as *npm* does. Results may also differ outside of an open-source context, e.g., when using badges to advertise practices among corporate teams.

¹In most cases, code is developed together in one repository, but deployed as multiple packages to reduce user download size when only parts of the project are needed.

²We considered all GITHUB-supported filename extensions for markdown files (see github.com/github/markup), most commonly README.md.

Table 1: Categories of badges present in our data.

Badge	Name	Description	Adoption	ST
QUALITY ASSURANCE				
	Travis CI	Build status	92789 (31.5%)	A
	Coveralls	Test coverage	17603 (6.0%)	A
	CodeClimate	Coverage & static analysis	6652 (2.3%)	A
	CodeCov	Test coverage	4788 (1.6%)	A
	Circle CI	Build status	3518 (1.2%)	A
	AppVeyor	Build status	2629 (0.9%)	A
	BitHound	Static analysis & dep. mgmt	1181 (0.4%)	A
	SauceLabs	Cross-browser testing	751 (0.3%)	A
	Inch CI	Documentation	437 (0.1%)	A
DEPENDENCY MANAGEMENT				
	David DM	Version tracking	23601 (8.0%)	A
	Gemnasium	Version tracking	2851 (1.0%)	A
	Greenkeeper	Version tracking	1599 (0.5%)	A
	Snyk	Vulnerability tracking	883 (0.3%)	A
	VersionEye	Version & vuln. tracking	240 (0.1%)	A
INFORMATION				
	Version	<i>npm</i> /GitHub version	64200 (21.8%)	L
	License	License information	6250 (2.1%)	S
	JS Standard	Coding style	5299 (1.8%)	S
	Semantic Rel.	Release strategy	1001 (0.3%)	S
	Commitizen	Commit msg. conventions	705 (0.2%)	S
	Heroku	Installation help	463 (0.2%)	S
	PRs Welcome	Static information	299 (0.1%)	S
POPULARITY				
	Downloads	<i>npm</i> download statistics	15552 (5.3%)	L
	cdnjs	Host of popular libraries	1902 (0.6%)	L
	Twitter	Twitter link and stats	810 (0.3%)	S
	GitHub Stars	Github statistics	630 (0.2%)	L
SUPPORT				
	Gitter	Chat & collaboration	4786 (1.6%)	S
	GitHub Issues	Issue statistics	1213 (0.4%)	L
	Slack	Chat & collaboration	688 (0.2%)	S
OTHER				
	Donation link	PayPal, Patreon, ...	1632 (0.6%)	S
	Donation stats	Gratipay, Gittip, ...	919 (0.3%)	L
	Ember Observer	Reviews and scoring	474 (0.2%)	A

ST (signal type): A—assessment signal based on nontrivial analysis or aggregation;
L—lookup of readily available information; S—static statement of information

3.2 Survey Insights and Hypotheses

Maintainer respondents to our survey generally see badges as important and a vast majority (88 %) agree with the statement “I consider the presence of badges in general to be an indicator of project quality.” Among contributors, badges were seen as more controversial: only 53 % agreed with the same statement and 61 % stated that badges do not influence their decision to contribute to a package. In their explanations some contributors indicated that badges pale in comparison to other reasons for contribution, but many also say that they consider them or expect to be influenced unconsciously.

Answers from both groups covered a spectrum of badges, most commonly build status, downloads, latest version on *npm*, and test coverage, but many others were also mentioned. We group badges in the following categories (see also Table 1): *quality assurance*, *dependency management*, *information*, *popularity*, *support*, and *other*. Corresponding to the distinction in signaling theory (§2), in Table 1 we distinguish badges with different signaling types [24] depending on whether they show results of deeper analyses (assessment signal), summarize readily available information, or merely state unvalidated information (both conventional signals).

Most importantly, the survey provides insights into what maintainers intend to signal and what consequences they expect, as well as insights into how contributors and users might interpret badges:

Quality assurance. All surveyed maintainers and most contributors (89 %) mentioned specific badges related to quality assurance. Most maintainers (84 %) stated explicitly that they intend to signal code and development quality—from “having any tests and running them regularly,” to signaling good quality assurance practices more generally, including striving for high coverage, standardizing code layout, and using static analysis tools. Respondents often intended to signal quality broadly, *e.g.*, stating that their badges show that their code was “built with love” or “well written” by an “experienced developer” who pays “attention to quality.”

H₁. *The adoption of quality-assurance badges correlates with other indicators of code quality (metric: test suite size).*

H₂. *The adoption of quality-assurance badges correlates with increased user confidence and attractiveness (metric: downloads).*

Several maintainers (28 %) also indicated that continuous integration, test coverage, and static analysis badges set expectations of contribution quality for new contributors. As one maintainer phrases it, with a coverage badge, “PRs with new functionality tend to include new tests, as not to decrease coverage.” This suggests coverage badges may have an additional gamification effect relative to other quality assurance badges.

H₃. *The adoption of a quality-assurance badge, and even more so of a coverage badge, correlates with more external contributors including tests (metric: percentage of PRs with tests).*

Dependency management. Dependency-management badges (mentioned in 26 % of all responses) indicate whether direct and indirect dependencies refer to outdated versions or even versions with known vulnerabilities. Respondents indicate that good dependency management practices (signaled with these badges) reduce the chance of “conflicting versions of nested dependencies” and indicate attention to updates and security patches.

H₄. *The adoption of dependency-management badges correlates with fresher dependencies (metric: freshness, see below).*

Information and navigation links. Though mentioned by many (56 %), usually as convenient shortcuts, we expect that link-related badges do not provide much signaling impact beyond other statements or links in the package description. Respondents suggested the following potential effects: Showing the latest *npm* version of a package as a badge might encourage more users to more quickly update to the latest version. Badges linking to *npm*, Heroku, CDNJS, or other external sites make it convenient for users to download and experiment with a package and may thus attract more users. Badges indicating high code quality (including style conventions) might encourage more users to attempt to read the code. Badges indicating explicitly that the package is open for contributions may lower the bar for new contributors. Badges indicating licenses can make it easier for users to make adopting decisions. Nonetheless, following signaling theory, we expect at most marginal effects.

H₅. *The adoption of a link-related badge does not correlate with either popularity or code quality.*

Popularity. Popularity badges were mentioned in 25 % of responses. Five maintainers explicitly mentioned that they intend to signal package popularity, to “instill confidence in new users,” and several contributors indicated that popularity is an important signal when deciding between similar packages, because the “wisdom of

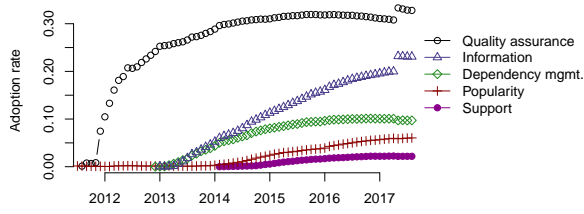


Figure 1: Badge adoption rate, by category; showing % of all *npm* packages in a given month with 1+ badge of that type.

the crowd” has deemed a package safe to use or of high quality. In addition, popularity is seen as a sign for likely sustainability. Especially the *npm*-downloads badge was mentioned frequently as an important addition to GitHub signals like stars and number of contributors. While downloads can be looked up on *npm*, a badge was often mentioned as a more convenient and direct means.

H₆. *The adoption of popularity-related badges in popular packages correlates with more future downloads (metric: monthly downloads).*

Support. Support badges were mentioned infrequently (9%), but some interpret them to signal “dedication to offering support.”

H₇. *The adoption of a support-related badges correlates with more responsive support (metric: issue closing time).*

Too many badges. An interesting facet to explore that came up a few times in the survey is that packages with too many badges can be perceived as cluttered or “trying too hard” and may be taken less seriously: “People tend to overwhelm visitors with too many (useless) badges, thus creating a contra effect and loosing the initial purpose of having useful information.” Hence our final hypothesis:

H₈. *The number of badges correlates non-linearly with popularity.*

3.3 Badge Popularity and Adoption

Of our 294,941 *npm* packages, 46% have at least one badge. Adoption statistics per badge (Table 1 for the most popular) reveal that **only few badges are broadly adopted**. Build status and version badges are by far the most common, followed by dependency managers, test coverage, and download statistics. A longitudinal analysis of badge adoption (Fig. 1) shows that quality-assurance badges (TRAVIS CI primarily) were adopted early and quickly, but seemed to have reached saturation (roughly every third new package adds a quality-assurance badge). Other kinds of badges have been adopted later and at lower rates; most seem to have reached saturation as well.

Badges tend to be adopted in groups and are not frequently changed afterward. Of 136,865 packages with badges, 66% adopted multiple kinds, of which 82% did so within 24 hours. Combinations of badges follow their overall popularity and typically involve quality-assurance, information, and dependency-manager badges. While there are often multiple badge-related commits when badges are first adopted (including temporary removal), only 13% of packages changed any badge more than 15 days after adopting their first. Permanent badge removal is also rare (11% of packages).

4 RQ₂: EFFECTS OF BADGES

After providing an overview of badge adoption in practice and collecting hypotheses about what their effects might be, we can

now test these hypotheses. In particular, we want to test to what degree the presence of badges correlates with certain expected qualities of the package, which we operationalize with measures such as downloads or rate of external contributions with tests.

4.1 Data and Methods

Data analysis. To evaluate the badges’ signaling reliability we proceed in three complementary steps per hypothesis.

Step 1: Correlation. We look for correlations between presence of badges and differences in the quality they are signaling. This analysis takes the outsider’s perspective of somebody looking at a repository now, and explores whether badges are reliable signals for certain qualities, independent of causal relationships, confounds, or historic trends. In line with our hypotheses, we typically analyze categories of badges together, as badges within a category can be expected to represent similar signals. We use the non-parametric WMW test to compare distributions and report Cliff’s delta.

Step 2: Additional information. Badges may correlate with various qualities, but still be redundant or weaker predictors of those qualities, compared to other signals. Here we explore whether badges *add* information to explain the qualities beyond readily-available signals, e.g., stars and issues shown on GitHub, downloads and dependent packages shown on *npm*. To assess the information gain with badges, we model the variability in the underlying quality using hierarchical linear regression. Specifically, we compare the fit of a *base model*, which includes only readily available signals and control variables, and a *full model*, which adds badge predictors. The difference, i.e., the added explanatory power attributable to badges, suggests their association with deeper-level qualities.

We use different types of linear models depending on the response variable (details with each result below), but always follow the same procedure for model fit and diagnostics. First, we conservatively remove outliers for predictors with exponential distributions, i.e., those values exceeding $k(1 + 2/n)\text{median}(x) + \theta$ [43], where θ is the exponential parameter [47], and k is computed such that not more than 1% of values are labeled as outliers; among these are high leverage points that disproportionately affect regression slopes, affecting robustness. Second, we diagnose the models, checking for multicollinearity (variance inflation factor, or VIF [2] below 3, except between the interaction terms and their comprising factors, which is expected), and checking if modeling assumptions hold (no significant deviation from a normal distribution in QQ-plots, randomly distributed residuals across the range). Finally, we consider model coefficients important if they are statistically significant at 0.05 level, and we estimate their effect sizes from ANOVA analyses.

Step 3: Longitudinal analysis. Previous steps look at differences between packages that have badges now. A longitudinal analysis could reveal whether packages with badges evolve differently than without, and *whether introducing a first badge (the intervention) has an observable effect on the package’s quality as the package evolves*. Here we use a powerful time series analysis method—*time series regression discontinuity design (RDD)* [18]—to evaluate longitudinal effects of displaying the first badge. With RDD, we estimate the magnitude of a function’s discontinuity between its values at points just before and just after an intervention. RDD is based on the assumption that in the absence of an effect, the function’s trend after the intervention

would be continuous in the same way as prior to the intervention. We consider the earliest display of a badge as the intervention, and compare data about the signaled underlying qualities in 18 monthly windows, 9 months on each side, centered around the adoption month. Aligning the history on the intervention date, we can compare 9-month trends before/after an intervention across many packages, looking for sudden jumps at the intervention and long-term differences in trends. We use multiple regression to estimate the trend in the response before the badge adoption (variable *time* in our models, e.g., Table 2), and the changes in level (variable *intervention*) and trend (variable *time_after_intervention*) after the badge adoption, cf. [65]. By controlling for confounds in the multiple regression (including presence of other badge classes), we evaluate whether the change could be attributed to other factors than the intervention.

To account for projects that adopt multiple badges (§3.3), we align on the first badge adoption but add controls for the adoption of other badges if they occur within 15 days of the first badge adoption. As usual, the controls allow us to isolate the effects of individual badges. The intuition is that, given the resolution of our analysis, interventions within 15 days can be considered as simultaneous. Later adoptions are not considered (existing only in 17,217 packages), but also do not systematically bias any specific month.

Repository mining and operationalization. Using the same dataset (294,941 packages; §3.1), we collected, besides badge adoption, data from three sources, both for a current snapshot for Steps 1 and 2 (July 2017) and longitudinally for Step 3 (monthly, January 2010–July 2017): (1) package metadata on *npm* using the *npm* API (e.g., downloads, releases, dependencies), (2) GitHub project data using the GitHub API and GHTORRENT [26] (e.g., contributors, issues, pull requests), and (3) the package’s git repository, cloned locally (e.g., code size, badges, tests). Specifically, we collected the following data. Representing readily-available signals of an *npm* package, we collected download statistics (*npm*), the number of stars and issues (GitHub), commit counts (git), and the size of the README file in bytes (git). As further controls, we collected the package’s age (*npm*), the size of the code base in bytes (git), the number of dependencies per package (*npm*), and the number of dependents, i.e., other packages depending on the package (*npm*).

We operationalize the qualities in our hypotheses with the following metrics for which we collect both current and historic values:

- Indicator of code quality (**H₁**): We measure the (relative) size of the test suite, by identifying files and directories that refer to test code, reusing the detector maintained by the package search service *npmjs.io*. We measure size in bytes, as it is robust to different test frameworks and file formats.
- Indicator of contribution quality (**H₃**): We measure how many pull requests contain test code (GitHub), using the same mechanism to identify test-related files.
- Indicator of users and popularity (**H₂**, **H₆**, **H₅**, **H₈**): We collect download counts from *npm*.
- Indicator of fresh dependencies (**H₄**): Inspired by recent work [19], we design a *freshness score* that performs a similar analysis to dependency managers, based on how many dependencies declared in a package have a newer version that existed on *npm* at the time (git, *npm*). For each package, we compute the average freshness of all direct dependencies, i.e., the average distance

between the specified version and the most recent release. Assuming that larger version changes require more effort to update, we assign a distance of 1 for each change of a patch version, 5 per minor version change, and 20 per major version change. An up-to-date dependency has distance 0. Details of the metric, including how version ranges are handled, are described in the appendix (Sec. C).

- Indicator of support (**H₇**): We collect the average time between when an issue is first posted and when it is closed, for all GitHub issues closed in July 2017, as the package’s average issue latency.

Threats to validity. *Imperfect measures.* Our operationalized measures can only capture some aspect of the underlying quality indicated in the survey. For example, large test suites are an indicator of good testing practices, but neither the only nor the most reliable indicator. However, while individual packages may vary in their practices, the large size of the data sets we study implies a reduction to the mean in terms of individual behavior. Therefore, we expect that by averaging over thousands of packages in our regression models, our (imperfect) measures will meaningfully reflect the intensity and directionality of underlying relationships between badges and project qualities.

To ensure internal validity, in our analysis Steps 2 and 3, we remove outliers and explore different operationalizations, whenever practical, to improve robustness. We further account for many covariates, especially other readily available signals, but we may miss other confounds easily observable by humans but hard to assess automatically, e.g., the quality of the documentation beyond our proxy of README file size. Consequently, one must be careful when generalizing our results beyond the studied measures.

Badges vs. practices. Typically, we cannot distinguish effects of practice adoption from effects of badge adoption; hence, our results can only be interpreted as exploring the reliability of the signal that a badge provides. Our analysis also does not consider the specific value shown on the badge (e.g., current coverage). Still, we expect that badges are usually adopted to signal good practices, a badge highlighting that a practice is not followed (e.g., low coverage) might have a negative effect. We control for this indirectly in many models, e.g., by controlling for popularity in our analysis of downloads (§4.3), but a more detailed analysis is outside the scope of this paper.

Beyond correlations. None of our three analysis steps can establish a causal relationship between badges and the studied qualities. Still, note how our three steps investigate each hypothesis from complementary perspectives: Step 1 checks for basic correlations, Step 2 explores the role of covariates and whether badges provide additional insights, and Step 3 looks at whether adopting badges leads to observable differences over time. The careful multi-faceted analysis, spearheaded by the sophisticated time-series regression discontinuity design, can indicate that correlations are not spurious and associate with some underlying phenomenon.

4.2 Signals of Updated Dependencies (**H₄**, **H₅**)

We explore our hypotheses grouped by response variable and start with a discussion of dependency freshness, as it clearly illustrates our 3-step analysis. We expect that dependency-manager badges

Table 2: Dependency freshness models.

	Basic Model response: <i>freshness</i> = 0 17.3% deviance explained		Full Model response: <i>freshness</i> = 0 17.4% deviance explained		RDD response: $\log(\text{freshness})$ $R_m^2 = 0.04, R_c^2 = 0.35$	
	Coeffs (Err.)	LR Chisq	Coeffs (Err.)	LR Chisq	Coeffs (Err.)	Sum sq.
(Intercept)	3.54 (0.03)***		3.50 (0.03)***		1.45 (0.09)***	
Dep.	-1.78 (0.01)***	32077.8***	-1.79 (0.01)***	32292.8***	-0.04 (0.02)	3.01
RDep.	0.22 (0.01)***	610.3***	0.21 (0.01)***	560.6***	-0.01 (0.02)	0.11
Stars	-0.08 (0.00)***	301.4***	-0.09 (0.00)***	311.2***	0.00 (0.01)	0.00
Contr.	-0.24 (0.01)***	500.5***	-0.25 (0.01)***	548.7***	-0.04 (0.02)*	4.39*
lastU	-0.65 (0.01)***	12080.9***	-0.64 (0.01)***	11537.9***	0.01 (0.02)	0.37
hasDM			0.24 (0.03)***	116.1***	0.45 (0.08)***	2.43
hasInf			0.11 (0.02)***	48.3***	0.04 (0.05)	0.45
hasDM:hasInf			-0.05 (0.04)	1.9	-0.32 (0.10)**	
hasOther			0.01 (0.01)			
time					0.03 (0.00)***	82.99***
intervention					-0.93 (0.03)***	1373.22***
time_after_intervention					0.11 (0.00)***	455.56***
time_after_intervention:hasDM					-0.10 (0.01)***	230.36***
time_after_intervention:hasInf					-0.00 (0.01)	1.14
time_after_intervention:hasDM:hasInf					0.03 (0.01)**	10.62**

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

Dep: dependencies; RDep: dependents; Contr.: contributors; lastU: time since last update; hasDM: has dependency-manager badge; hasInf: has information badge; hasOther: adopts additional badges within 15 days

correlate with more up-to-date and secure dependencies (**H₄**), operationalized with our *freshness metric* (see Sec. 4.1), and expect at most a marginal effect from information-related badges (**H₅**).

Correlation. Among the analyzed packages that had any dependencies, 37 % had all up-to-date dependencies (*freshness* = 0). Supporting **H₄** and, surprisingly, contradicting **H₅**, Fig. 2a reveals a small, but statistically significant difference: **packages with a dependency-manager badge or an information badge tend to have overall fresher dependencies than packages without.** We also find that dependency-manager badges are overproportionally adopted by packages with more dependencies.

Additional information. To test if the presence of badges correlates with deeper-level freshness indicators beyond other readily available signals, we fit a hurdle regression: a logistic model of the likelihood of *freshness* = 0 and a linear regression to model levels of freshness for packages with outdated dependencies. This hybrid modeling approach is necessary due to the bimodality of the data (Fig. 2a). As described in §4.1, the *base model* attempts to explain freshness given readily-available signals (stars, dependents, dependencies, contributors) and a control for *time since package was last updated*; the *full model* additionally models the presence of dependency-manager badges and information badges and their interaction, with controls for other badges adopted within 15 days.

We show the base and full logistic regression model (predicting whether a package has any outdated dependencies) in Table 2. The base model explains 17.3 % of the deviance; the full model explains 17.4 %. The difference is small but statistically significant (DeLong’s test for correlated ROC curves $p < 0.001$). The number of dependencies and the time since the last update explain the majority of the deviance, but **dependency-manager badges add explanatory power:** the odds of having fresh dependencies increase by 27 % ($e^{0.24}$) for packages with dependency-manager badges (**H₄**). Surprisingly, **the effect of information badges is comparable:** a 17 % increase in odds (**H₅**). For the linear regression (predicting the severity of outdated dependencies for packages with outdated dependencies), we see a similar small but significant difference between base (22.4 %) and full models (22.8 %), and similar behavior of the badge predictors.

Longitudinal analysis. We collect a sample of 3,604 packages that had dependencies and satisfy the RDD requirements (9 months activity before and after the adoption of their first dependency-manager badge), and keep 1,761 that had at least one month with *freshness* $\neq 0$ during the +/- 9 (to avoid data bimodality issues). A trend is already visible from the longitudinal freshness data plotted for those packages in Fig. 3a, but a corresponding RDD model controlling for confounds (column RDD³ in Table 2) confirms that: The adoption of (any) badges correlates to a strong improvement in freshness (see the *intervention* term in the model), by about a factor 2.5 on average,⁴ after which freshness slightly decays again over time (the interpretation derives from the sum of the coefficients for *time* and *time after intervention* in the model, cf. RDD [65], which expresses the slope of the post-intervention trend). As hypothesized, **the adoption of a dependency-manager badge is associated with a longer-lasting effect on freshness than other badges** (see the interaction *time after intervention* * *hasDM* in the model; ≈ 80 % slower decay). The interaction effect of information badges is negligible.

Discussion. Results from all three steps confirm **H₄** that dependency-manager badges signal practices that lead to fresher dependencies. However, the effect is not exclusive to dependency-manager badges; we speculate that any maintenance task involving README updates with more badges might involve other project cleanup. Still, the effect of dependency-manager badges is both stronger and longer lived, as signaling theory would predict given the assessment-signal nature of dependency-manager badges. The results are similar for a security score that counts known vulnerabilities, similar to the *Snyk* and *nsp* services (not shown due to space restrictions).

4.3 Signals of Popularity (**H₂**, **H₅**, **H₆**, **H₈**)

We expect that adopting quality-assurance and popularity badges correlates with increases in downloads (**H₂**, **H₆**), and at most a marginal effect from information-related badges (**H₅**). We follow the same three steps, analyzing monthly download counts as response.

Correlation. Supporting our hypotheses, comparing downloads for packages with and without badges shown in Figure 2b, we can see that for all categories of badges, **those packages with a badge tend to skew toward more downloads than packages without.** The differences are generally small, but statistically significant. As hypothesized, the effect for information badges is smaller than for quality assurance and popularity badges.

Additional information. We fit nested negative binomial regression models to explain the information added by badges for explaining downloads over readily-available signals and controls. Since already popular packages might benefit from badges in a different way than less popular ones, we also model a dummy variable *isPopular*, that indicates whether the package was among the 10 % most downloaded packages in the prior month (June 2017). In the model with badges, we then explore interactions with the *isPopular* dummy. Both models (see Sec. D) explain downloads well: The basic model explains 66 % of the deviance (with expected behavior of all main predictors), and the full model explains 86 %. The difference

³Note that all packages modeled in the RDD adopted some badge during the alignment month, hence the control *hasOther* is subsumed by experimental design.

⁴ $e^{0.93}$ factor decrease in freshness score; note the log-transformed response, hence the exponentiation here.

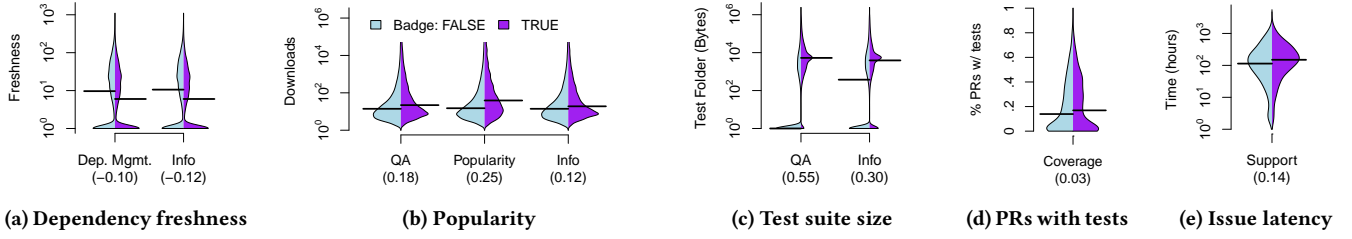


Figure 2: Distributions of response variables w/o and w/ badges. Horizontal lines depict medians. Cliff's delta below each plot.

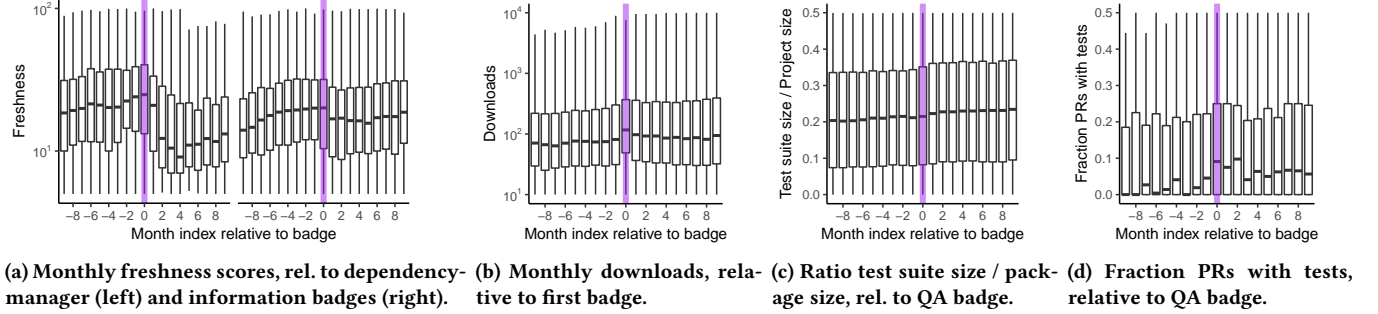
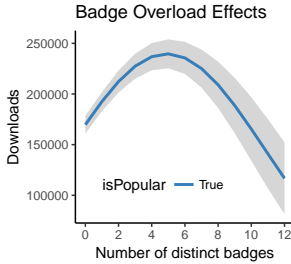


Figure 3: Trends in response variables before and after badge adoption.

is statistically significant; all badge categories have significant effects. Modeling interactions reveals that quality-assurance badges have a stronger effect in already popular packages: holding other variables constant at their mean values, popular packages with a quality-assurance badge tend to have about 2.2 times more downloads than comparable packages without; the effect is marginal for less popular packages. For popularity badges and information badges, we see smaller effects on popular packages (1.2 \times more, and 0.78 \times fewer downloads respectively). Again, the models show that badges explain additional aspects of popularity.

Separately, we also fit a negative binomial regression to model the effect of the number of badges displayed (and controls). The model (see Sec. D) suggests a nonlinear relationship in popular packages, with a predicted inflection point at 5 badges, which supports **H₈**: Packages with many badges tend to have fewer downloads. The effect for less popular packages is negligible.

Longitudinal analysis. We compile a set of 1,762 packages that satisfy the RDD requirements and have monthly download counts after March 2015 (a limitation of the *npm* API). Specifically, we align on the adoption month of their first badge for categories pertaining to our hypotheses. Of these, 1,414 packages adopted a quality-assurance badge, 892 an information badge, and 366 a popularity badge. Prior to modeling, we *inflation-adjusted* each package's monthly download counts to account for *npm*'s natural growth over time, based on the average download growth of 10,000 randomly-sampled packages with at least 10 dependents each, that existed the entire period.



A visual inspection of trends around the first badge adoption (Fig. 3b) shows potential intervention effects; an RDD model controlling for confounds (see Sec. D) suggests a small positive trend prior to the intervention, a sizeable positive discontinuity in download counts at badge adoption (33% increase on average), and, surprisingly, a small negative slope after the intervention. That is, badge adoption correlates with a sudden popularity boost, but the acceleration is not sustained over time. The post-intervention decay for quality-assurance badges is slower than average (8.8%)—in other words, the positive intervention effect lasts longer.

Discussion. Together, our three analysis steps paint a mixed picture for our hypotheses: All badges correlate with more downloads in general, and a longitudinal analysis shows also positive intervention effects for quality-assurance and popularity badges (**H₂**, **H₅**, **H₆**), but the pace is not sustained over time. Still, not all packages with badges show similar effects: Those with assessment-signal badges (namely quality-assurance) tend to maintain the popularity boost that correlates with badge adoption longer, as signaling theory would predict. At the same time, too many badges may seem counterproductive (**H₈**).

4.4 Signals of Test Suite Quality (**H₁**, **H₅**)

We expect that the adoption of quality-assurance badges correlates with increases in test-suite quality (**H₁**), operationalized as the size of the test suite, and again at most a marginal effect from information badges (**H₅**). We follow our common 3-step analysis.

Correlation. The distribution of test suite sizes across packages is bimodal (Figure 2c), as we cannot detect any test code in 39.3% of the packages. Packages with quality-assurance badges almost always had tests (93.5%). Among the packages with tests, those packages with quality-assurance badges tend to skew toward larger

test suites, with statistically significant differences, supporting our hypothesis. Surprisingly, so do packages with information badges.

Additional information. Given the bimodal distribution of test suite sizes, as for dependency freshness (Sec. 4.2), we fit a hurdle regression (see Sec. E), modeling separately the likelihood of having any tests (logistic regression) and the test-folder size for packages with non-empty test suites (negative binomial regression). In addition to the usual controls, we control for the *package size* (larger packages are expected to have larger test suites).

The base logistic model is plausible for explaining whether a package has any tests (4.4 % deviance explained). The full model with quality-assurance and information badges fits the data significantly better (22.6 % deviance explained), most being attributed to quality-assurance badges. The full negative binomial regression model shows a small improvement for explaining test-suite size (78.2 % to 78.4 %). In both cases (having any tests and size of test suite), the models show a strong positive effect of quality-assurance badges (H_1): on average, the odds of having tests increase by a factor 18 for packages with quality-assurance badges; among packages with tests, those with quality-assurance badges are expected to have 18.3 % larger test suites, other variables held constant. Information badges have a marginal effect on their own but interact with quality-assurance badges, strengthening their effect slightly (H_5).

Longitudinal analysis. To avoid technical problems with the bimodal distribution, we study only how first badge adoption correlates with the growth of an existing test suite, not whether it coincided with creating a test suite in the first place. To this end, we assemble a longitudinal sample of 2,855 packages that adopted a quality-assurance badge and had at least 9 months of history with non-empty test suites, both before and after the adoption month. An RDD model controlling for confounds (see Sec. E) reveals what is barely visible in the plotted data (Fig. 3c): there is a small but statistically significant positive shift in test-suite size at the intervention, but almost no change in slope after the badge adoption. Stated differently, introducing quality-assurance badges coincides with an improvement in the test suite, but does not trigger a lasting change to testing practices. Information badges are correlated with a much smaller effect.

Discussion. As expected by the surveyed maintainers, all three steps indicate that quality-assurance badges are a good signal for a project having some tests, though they are a weaker signal for the size of the test suite (H_1). Since the adoption of quality-assurance badges is correlated with an intervention effect on test suite size, we conclude that they may act as gamification mechanisms. Again, we see a marginal change at the intervention for information badges; the former can likely be explained through some overall perception of well-managed projects (H_5).

4.5 Signals of Better Contributions (H_3)

We expect that adopting quality assurance, and especially coverage badges, correlates with better pull requests (H_3), which we operationalize as the likelihood of pull requests containing tests.

Correlation. A direct comparison (Fig. 2d) shows that, supporting our hypothesis, packages with quality-assurance badges tend to have slightly higher fractions of pull requests with tests. Note

that coverage badges are adopted almost exclusively together with continuous integration badges (97 %), but not vice versa (23 %).

Additional information. We compile a set of 3,344 packages having more than 50 pull requests. Of these, 2,693 have at least one badge (944 coverage badge, 2289 other quality-assurance badge) and 651 have no badges. We then fit a nested logistic regression (with our standard controls; 12.9 % deviance explained in base model, 14.2 % in full; see Sec. F) to model the fraction of pull requests per package containing tests. We find that non-coverage quality-assurance badges generally have a positive effect, increasing the chance of tests in a pull request by 24.1 % (all other factors held constant); coverage and other quality assurance badges interact, amplifying each other's effects by an additional 16.8 % if a coverage badge and a continuous-integration badge are adopted together.

Longitudinal analysis. We analyze a sample of 324 packages with 9 months of history on each side of the coverage badge adoption month using our standard RDD approach. Visually (Fig. 3d), the pre and post-badge periods are clearly distinguishable, with the post-badge period showing more pull requests with tests. The model (see Sec. F) confirms an increase in the monthly fraction of pull requests containing tests after adopting quality-assurance badges (on average 23 %), and a slow decay afterward that is further slowed (5.7 %) if coverage badges are adopted additionally.

Discussion. All three steps support H_3 : Packages with quality-assurance badges receive better pull request contributions; coverage badges tend to amplify this effect, even increasing the duration of the intervention effect, which supports our hypothesis that they are gamification mechanisms, inspiring contributors to include tests in order to not decrease the displayed coverage percentage.

4.6 Signals of Support (H_7)

We expect that the adoption of support badges correlates with better support (H_7), operationalized as the average time to close issues on GitHub.

Unfortunately, there are relatively few packages that have adopted support badges (see Tab. 1). We limited our analysis to 826 packages with at least 100 issues overall, and having at least one issue created after July 1, 2017 and closed before Aug. 20; 397 of these packages have support badges. In contrast to our hypothesis, we observe that projects with a support badge have, on average, 20 % longer closing times than those without; however, we also observe that projects with support badges receive twice the number of issues on average. When controlling for confounds, including the number of issues (base model 21.2 %, full model 21.3 %), we find a similar effect: Projects with support badges have a 30.2 % higher latency on average. Removing packages that do not fulfill the RDD constraints leaves only 76 packages. Controlling for the same confounds, the RDD model only suggests that issue latency decreases with time, but does not show an effect of support badges.

Overall, our findings for hypothesis H_7 are negative and opposite to our expectations. The results in Steps 1 and 2 are statistically significant, but all steps suffer from small data sets. We have no explanation for this effect beyond conjecturing that external support platforms (GITTER and SLACK) handle support requests in a way that is not captured by our operationalization with GitHub issues.

5 DISCUSSION AND CONCLUSION

We studied repository badges, a new phenomenon in social coding environments like GITHUB, with previously unknown effects.

Research questions. We answered two research questions. First, exploring the most common types of badges and their intended signals (RQ₁), we found a diversity of badges and signals (Table 1): some are merely static displays of (existing) information, others aggregate information that is otherwise much harder to observe, e.g., reflecting build status, up-to-dateness of dependencies, and test coverage. As predicted by signaling theory (§2), our survey revealed that package maintainers displaying badges have clear signaling intentions doing so, and many contributors interpret badges when evaluating packages. Second, exploring the fit of the signals to underlying qualities hypothesized by our survey participants (RQ₂), we found that packages with badges tend to skew towards having more of the quality they signal, with stronger effects for the non-trivial quality-assurance and dependency-manager badges. Moreover, the presence of badges consistently adds explanatory power, albeit little, to readily-available signals. Time-series analysis further revealed that the introduction of quality-assurance badges tends to correlate with positive intervention effects: The underlying qualities they signal tend to improve immediately, especially improved dependency freshness and more tests in pull requests.

Gamification effects. Our results also revealed gamification effects (§2): Clear examples are the quality assurance badges displaying test coverage percentages (§4.5), which we found to correlate with developers increasing the size and, arguably, quality of their test suites. Dependency-management badges (§4.2) could also be seen as a gamification mechanism: by making the up-to-dateness of dependencies noticeable, they create an incentive for developers to make the most out of their dependency-management tools by staying up to date. One can imagine other badges with gamification value, e.g., around bug fixing, being used in the future to encourage desirable practices. However, we advocate caution in implementing gamification and acknowledge the risk of creating the wrong incentives, e.g., writing tests only to maximize coverage.

The power of assessment signals. As discussed in §2, signaling theory makes an interesting distinction between conventional signals and assessment signals, where the latter require that the signaler actually possesses the signaled quality. Although a few survey participants suggested intended signals for information badges and support badges (§3.2), the theory predicts that, without an associated cost or analysis, they are less reliable signals. Our results seem to confirm that quality assurance, dependency manager, and popularity badges (mostly assessment signals) provide more reliable signals than information badges (mostly conventional signals). For example, the effect on freshness is stronger and longer lasting for the assessment signal. Interestingly, information badges often do signal something, even outside their domain of cost (in fact, in data exploration we often also found other small effects of other badges on various qualities). We speculate that badges are often adopted during a general maintenance phase in which also test suites or dependencies are improved, but we expect that only assessment signals correlate with lasting change (§4.2).

The results encourage more badges to be designed as assessment signals. For example, several badges that currently only state intentions, such as `code style standard` or `gitter join chat`, could be redesigned to report analysis result for the underlying quality, such as past conformance to coding standards or responsiveness to support requests. Such badges may encourage stronger conformance and even accrue gamification benefits. Interestingly, *Slack* offers multiple kinds of badges, including one inviting users to join (`slack join`) and one that shows the number of currently active and registered users (`slack 6/160`). In practice though, most package maintainers with a *Slack* badge adopt the former (conventional signal) rather than the latter (assessment signal). Our results indicate that maintainers, when they have the choice and are serious about signaling their dedication, should adopt the assessment-signal badge.

Badges vs practices. We emphasize that in most cases effects associated with non-trivial badges are inextricably linked to effects associated with the badges' corresponding tools or practices. For many kinds of badges, the badge is the only (easily) externally observable indicator of the tool's use (e.g., `dependencies up to date`, `docs`). While effects associated with adopting continuous integration [31, 60, 65], static analysis [7, 64], and dependency-management tools [41] on software development practices have been studied by prior work, our approach is unique in that we focus on the *signaling dimension* added by repository badges to these and other tools, previously overlooked in the software engineering signaling literature (e.g., [20, 38, 48, 55, 56]). Further delineating effects associated with badges from effects associated with the tools themselves goes beyond the scope of this work, but some preliminary experiments with the continuous integration service TRAVIS, which is detectable independently of its badge through a `travis.yml` configuration file in the repository, suggest that while the observed effects of adopting the tool are similar to the effects at the badge adoption time, badges seem to have a small amplifying effect. Pending further validation, this is encouraging for researchers, in that badges might be a reliable indicator for *longitudinal* studies of practices that are hard to detect otherwise, because there are no clear traces of the practice in the repository, beyond the badge.

Implications for practitioners. Our results provide guidance on which qualities are usually signaled with badges and which signals tend to be more reliable. Therefore, *package maintainers* can make more deliberate choices about badges (e.g., limiting conventional signals), *service developers* can design badges more carefully (e.g., providing an assessment signal based on some analysis of past conformance), and *package users and contributors* can decide which badges to use as indicators of underlying practices and as starting points to investigate deeper qualities. Overall: `signals mostly reliable`.

Acknowledgements. Many thanks to respondents to our survey! Trockman was supported through Carnegie Mellon's Research Experiences for Undergraduates in Software Engineering. Kästner and Zhou have been supported in part by the NSF (awards 1318808, 1552944, and 1717022) and AFRL and DARPA (FA8750-16-2-0042). Vasilescu has been supported in part by the NSF (award 1717415).

6 ADDITIONAL MATERIAL

Plots, models, and exact survey questions are available online at <https://github.com/CMUSTRUDEL/npm-badges>.

REFERENCES

- [1] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. 2017. Why Do Developers Use Trivial Packages? An Empirical Case Study on *npm*. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM.
- [2] Paul D Allison. 1999. *Multiple regression: A primer*. Pine Forge Press.
- [3] Bilal Amir and Paul Ralph. 2014. Proposing a theory of gamification effectiveness. In *Companion Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 626–627.
- [4] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. 2013. Steering user behavior with badges. In *Proc. Int'l Conf. World Wide Web (WWW)*. ACM, 95–106.
- [5] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2015. How the Apache community upgrades dependencies: An evolutionary study. *Empirical Software Engineering* 20, 5 (2015), 1275–1317.
- [6] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. 2013. Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. *IEEE Software* 30, 1 (2013), 52–66.
- [7] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. 2016. Analyzing the state of static analysis: A large-scale evaluation in open source software. In *Proc. Int'l Conf. Software Analysis, Evolution and Reengineering (SANER)*, Vol. 1. IEEE, 470–481.
- [8] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proc. Working Conf. Mining Software Repositories (MSR)*. ACM, 137–143.
- [9] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. 2007. Open borders? Immigration in open source projects. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 6.
- [10] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. 2016. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology* 70 (2016), 30–39.
- [11] 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. Int'l Symp. Foundations of Software Engineering (FSE)*. ACM, 109–120.
- [12] John Businge, Alexander Serebrenik, and Mark GJ van den Brand. 2015. Eclipse API usage: the good and the bad. *Software Quality Journal* 23, 1 (2015), 107–141.
- [13] Andrea Capiluppi, Alexander Serebrenik, and Leif Singer. 2013. Assessing technical candidates on the social web. *IEEE Software* 30, 1 (2013), 45–51.
- [14] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. 2015. Developer onboarding in GitHub: the role of prior social links and language experience. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 817–828.
- [15] Huseyin Cavusoglu, Zhuolun Li, and Ke-Wei Huang. 2015. Can gamification motivate voluntary contributions? The case of Stack Overflow Q&A community. In *Companion Proc. Conf. Computer Supported Cooperative Work & Social Computing (CSCW)*. ACM, 171–174.
- [16] Jailton Coelho and Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM.
- [17] Brian L Connelly, S Trevis Certo, R Duane Ireland, and Christopher R Reutzel. 2011. Signaling theory: A review and assessment. *Journal of Management* 37, 1 (2011), 39–67.
- [18] Thomas D Cook, Donald Thomas Campbell, and Arles Day. 1979. *Quasi-experimentation: Design & analysis issues for field settings*. Vol. 351. Houghton Mifflin Boston.
- [19] Joël Cox, Eric Bouwers, Marko van Eekelen, and Joost Visser. 2015. Measuring Dependency Freshness in Software Systems. In *Proc. Int'l Conf. Software Engineering, Volume 2*. IEEE Press, 109–118.
- [20] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. Conf. Computer Supported Cooperative Work (CSCW)*. ACM, 1277–1286.
- [21] Laura Dabbish, Colleen Stuart, Jason Tsay, and James Herbsleb. 2013. Leveraging transparency. *IEEE Software* 30, 1 (2013), 37–43.
- [22] Alexandre Decan, Tom Mens, and Maëlick Claes. 2017. An empirical comparison of dependency issues in OSS packaging ecosystems. In *Proc. Int'l Conf. Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2–12.
- [23] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. 2011. Gamification. using game-design elements in non-gaming contexts. In *Proc. Conf. Human Factors in Computing Systems (CHI), Extended Abstracts*. ACM, 2425–2428.
- [24] Judith Donath. 2007. Signals in social supernets. *Journal of Computer-Mediated Communication* 13, 1 (2007), 231–251.
- [25] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. 2017. Some from here, some from there: cross-project code reuse in GitHub. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 291–301.
- [26] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *Proc. Working Conf. Mining Software Repositories (MSR)*. ACM, 384–387.
- [27] Scott Grant and Buddy Betts. 2013. Encouraging user behaviour with achievements: an empirical study. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 65–68.
- [28] Tim Guilford and Marian Stamp Dawkins. 1991. Receiver psychology and the evolution of animal signals. *Animal Behaviour* 42, 1 (1991), 1–14.
- [29] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie Van Deursen. 2013. Communication in open source software development mailing lists. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 277–286.
- [30] Joseph Hejderup. 2015. *In dependencies we trust: How vulnerable are dependencies in software modules?* Master's thesis. TU Delft, The Netherlands.
- [31] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-source Projects. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. ACM, 426–437.
- [32] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE Press, 164–174.
- [33] Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. 2015. Open source-style collaborative development practices in commercial projects using GitHub. In *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 574–585.
- [34] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2017. Do developers update their library dependencies? *Empirical Software Engineering* (2017), 1–34.
- [35] Michael J Lee, Bruce Ferwerda, Junghong Choi, Jungpil Hahn, Jae Yun Moon, and Jinwoo Kim. 2013. GitHub developers use rockstars to overcome overflow of news. In *Proc. Conf. Human Factors in Computing Systems (CHI), Extended Abstracts*. ACM, 133–138.
- [36] M Lynne Markus and Brook Manville Carole E Agres. 2000. What makes a virtual organization work? *MIT Sloan Management Review* 42, 1 (2000), 13.
- [37] Jennifer Marlow and Laura Dabbish. 2013. Activity traces and signals in software developer recruitment and hiring. In *Proc. ACM Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 145–156.
- [38] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in GitHub. In *Proc. Conf. Computer Supported Cooperative Work (CSCW)*. ACM, 117–128.
- [39] Cade Metz. 2015. How GitHub Conquered Google, Microsoft, and Everyone Else. <https://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else/>. (2015).
- [40] Vishal Midha and Prashant Palvia. 2012. Factors affecting the success of Open Source Software. *Journal of Systems and Software* 85, 4 (2012), 895–905.
- [41] Samim Mirhoseini and Chris Parnin. 2017. Can Automated Pull Requests Encourage Software Developers to Upgrade Out-of-Date Dependencies?. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. to appear.
- [42] Audris Mockus, Roy T Fielding, and James D Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.
- [43] Jagdish K Patel, CH Kapadia, and Donald Bruce Owen. 1976. *Handbook of statistical distributions*. M. Dekker.
- [44] Steven Raemaekers, Arie van Deursen, and Joost Visser. 2014. Semantic Versioning versus Breaking Changes: A Study of the Maven Repository. In *Proc. Int'l Working Conf. Source Code Analysis and Manipulation (SCAM)*. IEEE Computer Society, 215–224.
- [45] Eric S Raymond. 2001. *The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary*. O'Reilly Media, Inc.
- [46] RightScale. 2016. State of the Cloud Report: DevOps Trends. <http://www.rightscale.com/blog/cloud-industry-insights/new-devops-trends-2016-state-cloud-survey>. (2016).
- [47] Peter J Rousseeuw and Christophe Croux. 1993. Alternatives to the median absolute deviation. *J. Amer. Statist. Assoc.* 88, 424 (1993), 1273–1283.
- [48] N Sadat Shami, Kate Ehrlich, Geri Gay, and Jeffrey T Hancock. 2009. Making sense of strangers' expertise from signals in digital artifacts. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 69–78.
- [49] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. 2014. Understanding watchers on GitHub. In *Proc. Working Conf. Mining Software Repositories (MSR)*. ACM, 336–339.
- [50] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software engineering at the speed of light: How developers stay current using Twitter. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 211–221.
- [51] Michael Spence. 1973. Job market signaling. *The Quarterly Journal of Economics* 87, 3 (1973), 355–374.
- [52] Michael Spence. 2002. Signaling in retrospect and the informational structure of markets. *The American Economic Review* 92, 3 (2002), 434–459.
- [53] Megan Squire. 2015. "Should We Move to Stack Overflow?" Measuring the Utility of Social Media for Developer Support. In *Proc. Int'l Conf. Software Engineering (ICSE)*, Vol. 2. IEEE, 219–228.

- [54] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (r)evolution of social media in software engineering. In *Proc. Workshop on the Future of Software Engineering*. ACM, 100–116.
- [55] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM, 356–366.
- [56] Jason Tsay, Laura Dabbish, and James D Herbsleb. 2013. Social media in transparent work environments. In *Proc. International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 65–72.
- [57] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2015. Perceptions of diversity on GitHub: A user survey. In *Proc. Int'l Workshop Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 50–56.
- [58] Bogdan Vasilescu, Alexander Serebrenik, Prem Devanbu, and Vladimir Filkov. 2014. How social Q&A sites are changing knowledge sharing in open source software communities. In *Proc. Conf. Computer Supported Cooperative Work & Social Computing (CSCW)*. ACM, 342–354.
- [59] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. 2015. A Data Set for Social Diversity Studies of GitHub Teams. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 514–517.
- [60] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE) (ESEC/FSE)*. IEEE, 805–816.
- [61] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. 2016. A look at the dynamics of the JavaScript package ecosystem. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 351–361.
- [62] Kazuhiro Yamashita, Yasutaka Kamei, Shane McIntosh, Ahmed E Hassan, and Naoyasu Ubayashi. 2016. Magnet or sticky? Measuring project characteristics from the perspective of developer attraction and retention. *Journal of Information Processing* 24, 2 (2016), 339–348.
- [63] Amotz Zahavi. 1975. Mate selection—a selection for a handicap. *Journal of Theoretical Biology* 53, 1 (1975), 205–214.
- [64] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How open source projects use static code analysis tools in continuous integration pipelines. In *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 334–344.
- [65] Yangyang Zhao, Yuming Zhou, Alexander Serebrenik, Vladimir Filkov, and Bogdan Vasilescu. 2016. The Impact of Continuous Integration on Other Software Development Practices: A Large-Scale Empirical Study. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. IEEE.

APPENDIX

B FOREWORD

Listed below are the models and survey questions used for the study. Effect plots are provided to assist in the interpretation of the longitudinal models.

The tables were created using the R package *stargazer* by Marek Hlavac.⁵

Our data set and R source code (used for the models in the paper and to generate this appendix) is publicly available.⁶

C SIGNALS OF UPDATED DEPENDENCIES

C.1 Freshness Scores

Given a list of releases for a *single* dependency, such as

(1.0.0, 1.0.1, 1.2.1, 2.0.1-alpha, 2.0.2)

we define the *freshness* to be a weighted distance from the current version to the newest version.

For example, if the current version is given by the package as 1.0.x, then the maximum version matching this in the list above is 1.0.1. Hence we are

1.0.1 \Rightarrow 1.2.1: one minor version (5 points)

1.2.1 \Rightarrow 2.0.1-alpha: one major version (20 points)

2.0.1-alpha \Rightarrow 2.0.2: one patch version (1 point)

= 26 points away from the latest version.

To handle the version strings used in package.json files, we use the *npm* packages *semver* and *semver-diff*. This allows us to find, for example, the latest release fulfilling a given version string. *semver* is used by *npm* itself.

We recognize that there are different “newest releases” depending on the time. To solve this, we created a database of releases for all *npm* packages and only considered those that occurred before the time under consideration.

The per-project freshness scores in the paper are averaged over all of the project’s dependencies.

⁵<https://cran.r-project.org/web/packages/stargazer/vignettes/stargazer.pdf>

⁶<https://github.com/CMUSTRUDEL/npm-badges>

Algorithm 1: Semantic version string utility functions

The following functions are provided by *npm* packages *semver*^a and *semver-diff*^b. We outline them here for convenience.

```

Function SemverSatisfies?(specific version, version range):
  if specific version in version range then
    | return True
  else
    | return False
Function SemverDiff(version1, version2):
  if version2 differs by a major version number from version1 then
    | /* e.g. 1.0.0 and 2.0.0 */
    | return major
  else if version2 differs by a minor version number from version1 then
    | /* e.g. 2.0.0 and 2.1.0 */
    | return minor
  else if version2 differs by a patch version number from version1 then
    | /* e.g. 3.5.6 and 3.5.8 */
    | return patch
  else if version2 < version1 or the difference cannot be determined then
    | /* e.g. 1.0.0-alpha1 and 1.0.0-alpha2 */
    | return error
  end

```

^a<https://github.com/npm/node-semver>^b<https://github.com/sindresorhus/semver-diff>**Algorithm 2:** Calculate the freshness score for a single dependency

Data: *R* is a list of releases sorted by date, e.g.
 [1.0.0, 1.0.1, 1.0.2, 1.1.2, 1.1.3, 1.2.0, 2.0.0, 2.0.1, 2.1.0]
v is the current version range string, e.g. 1.1.x
Result: Freshness score for a package at version *v* given its releases *R*

```

Function Freshness(R, v):
  matches ← array with same length as R
  for i ← 1 to length(R) do
    | matches[i] ← SemverSatisfies?(R[i], v)
  end
  if matches all false then
    | return null /* Invalid version string: error */
  end
  if R[length(R)] is True then
    | return 0 /* Entirely up-to-date. */
  end
  lastTrue ← max{i : matches[i] is True }
  score ← 0
  for i ← lastTrue to length(R) - 1 do
    versionDiff ← SemverDiff(R[i], R[i + 1])
    if versionDiff == major then
      | score ← score + 20
    else if versionDiff == minor then
      | score ← score + 5
    else if versionDiff == patch then
      | score ← score + 1
    else
      | /* Could not find difference, perhaps a non-standard version
      | string was encountered in R. Nonetheless, we know a release did
      | occur. */
      | score ← score + 1
    end
  end
  return score

```

Table 3: Dependency “freshness” (1-4) at the time of study and (5) per month

	<i>Dependent variable:</i>				
	log(freshness + 1)		(freshness == 0)		log(freshness + 1)
	<i>OLS</i>		<i>logistic</i>		<i>linear</i>
	Base Model		Base Model		<i>mixed-effects</i>
	Full Model		Full Model		RDD
	(1)	(2)	(3)	(4)	(5)
log(dependencies + 1)	−0.272***	−0.267***	−1.771***	−1.781***	−0.040*
log(dependents + 1)	−0.118***	−0.110***	0.216***	0.207***	−0.005
log(tsu + 1)	0.510***	0.503***	−0.658***	−0.651***	0.013
time					0.027***
intervention1					−0.929***
time_after_intervention					0.112***
hasDepmgr1					0.446***
hasInfo1					0.038
time_after_intervention:hasDepmgr1					−0.099***
time_after_intervention:hasInfo1					−0.004
hasDepmgr1:hasInfo1					−0.320***
time_after_intervention:hasDepmgr1:hasInfo1					0.035***
log(stars + 1)	0.061***	0.063***	−0.085***	−0.086***	−0.0001
log(contributors + 1)	0.122***	0.130***	−0.239***	−0.252***	−0.041**
hasExtra		−0.033***		0.011	
hasDepMgrBadge1		−0.103***		0.228***	
hasInfoBadge1		−0.075***		0.104***	
hasDepMgrBadge1:hasInfoBadge1		−0.031		−0.047	
Constant	2.061***	2.098***	3.568***	3.530***	1.452***
AIC	334670	334112	203930	203651	
Projects	293444	293444	293444	293444	1762(x19)
Adjusted R^2	0.224	0.228			
Pseudo- R^2			0.173	0.174	
R_m^2					0.0428
R_c^2					0.347

Note:

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

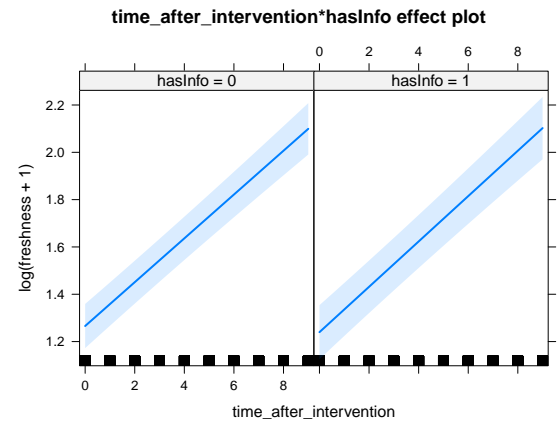
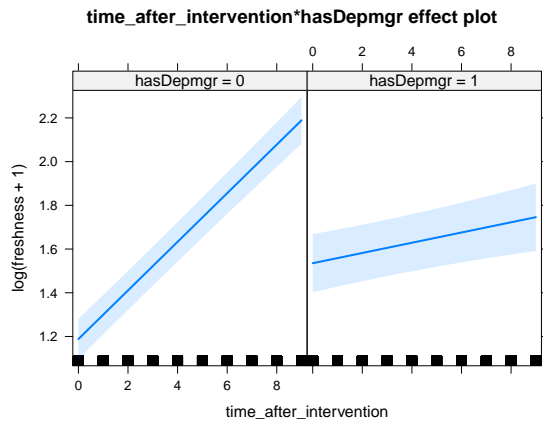
DeLong’s ROC Test. We test to see if there is a significant difference between the base and full logistic regression models for up-to-dateness/freshness using DeLong’s ROC test.

```
##
## DeLong's test for two correlated ROC curves
##
## data: fitted(mod.fresh.badge.hurdle) and fitted(mod.fresh.base.hurdle) by mod.fresh.badge.hurdle$y (0, 1)
## Z = -108.47, p-value < 2.2e-16
## alternative hypothesis: true difference in AUC is not equal to 0
## sample estimates:
## AUC of roc1 AUC of roc2
## 0.7742829 0.7735593
```

C.2 Effect Plots

The adoption of both dependency manager badges and information badges seems to be correlated with an increase in dependency freshness. However, the effect lasts longer for dependency manager bages, as can be seen below.

Recall that lower freshness is better (more up-to-date). Projects with dependency manager badges tend to stay at a similar level after the intervention, whereas the other classes correlate with decaying freshness (upward sloping).



D SIGNALS OF POPULARITY

Table 4: Download counts per (1-3) month of the study (4) month, as available

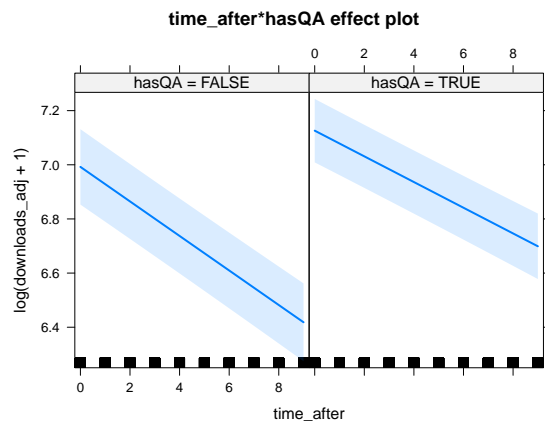
	<i>Dependent variable:</i>			
	downloads		log(downloads_adj + 1)	
	<i>negative binomial</i>		<i>linear mixed-effects</i>	
	Base Model	Full Model	Num. Badges	RDD
	(1)	(2)	(3)	(4)
log(age + 1)	-0.160***	-0.246***	-0.045***	
isPopular		4.954***		
log(size + 1)			-0.171***	
time				0.020***
time_after				-0.068***
intervention1				0.291***
log(stars + 1)	0.245***	0.121***	-0.092***	0.103***
log(num_issues + 1)	0.240***	0.079***		
log(revisions + 1)	0.432***	0.517***	-0.006	0.682***
log(dependents + 1)	3.178***	1.172***	1.422***	0.947***
log(dependencies + 1)				-0.015
log(readme_size + 1)	0.109***	0.064***	-0.099***	0.001
hasQABadge1		0.031***		
hasDepMgrBadge1		0.135***		
hasPopularityBadge1		0.135***		
hasInfoBadge1		-0.139***		
isPopularTRUE:hasQABadge1		0.820***		
isPopularTRUE:hasDepMgrBadge1		-0.480***		
isPopularTRUE:hasPopularityBadge1		-0.087***		
isPopularTRUE:hasInfoBadge1		-0.128***		
num_badges			0.141***	
num_badges_sq			-0.014***	
log(ght_age + 1)				0.544***
hasQA				0.134**
hasInfo1				-0.074
hasPopularity1				0.261***
hasOtherBadge				-0.032
time_after:hasQA				0.016***
time_after:hasInfo1				0.003
time_after:hasPopularity1				0.0003
time_after:hasOtherBadge				0.011***
Constant	3.418***	2.733***	11.113***	-0.301
AIC	3470725	3135433	650227	
Projects	294041	294041	294041	294041
Pseudo- R^2	0.655	0.86	0.524	
R_m^2				0.552
R_c^2				0.932

Note:

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

D.1 Effect Plots

We find that the intervention effect correlated with adopting a badge lasts longer if a quality assurance badge is adopted within 15 days of the intervention. In the effect plot below, the slope is smaller for hasQA = 1.



E SIGNALS OF TEST SUITE QUALITY

Table 5: Test suite size in bytes per project (1-4) at time of study (5) monthly

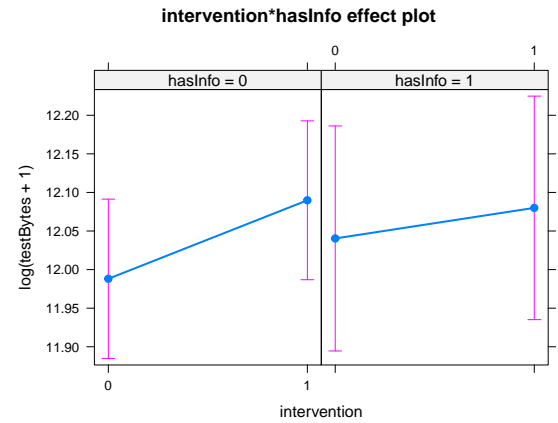
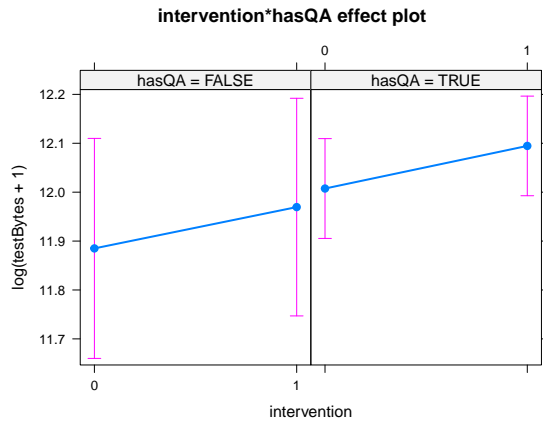
	<i>Dependent variable:</i>				
	test_bytes		(test_bytes >0)		log(testBytes + 1)
	<i>negative binomial</i>		<i>logistic</i>		<i>linear mixed-effects</i>
	Base Model	Full Model	Base Model	Full Model	RDD
	(1)	(2)	(3)	(4)	(5)
log(proj_bytes + 1)	0.957***	0.963***	0.183***	0.183***	
log(projBytes)					0.718***
log(stars + 1)	-0.008***	-0.015***	-0.027***	-0.079***	0.020
log(dependents + 1)	0.066***	0.060***	0.616***	0.501***	
log(revisions + 1)	-0.022***	-0.018***	0.044***	0.036***	0.113***
log(age + 1)	0.078***	0.106***	0.053***	0.188***	
hasQABadge		0.168***		2.898***	
hasInfoBadge1		-0.103***		-0.175***	
hasExtraBadge		-0.186***		-0.421***	
hasQABadge:hasInfoBadge1		0.184***		0.317***	
log(downloads + 1)					0.020**
time					0.014***
intervention1					0.087***
time_after_intervention					-0.006*
hasInfo1					0.052
hasQA					0.122
hasOther					-0.068
intervention1:hasInfo1					-0.062***
intervention1:hasQA					0.003
intervention1:hasOther					0.068***
Constant	-1.132***	-1.345***	-1.778***	-2.670***	0.738***
AIC	3628951	3627352	376165	304579	
Projects	293664	293664	293664	293664	2358(x19)
Pseudo- R^2	0.782	0.784	0.044	0.226	
R_m^2					0.575
R_c^2					0.955

Note:

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

E.1 Effect Plots

From the effect plots, we see that the adoption of a badge is correlated with an increase in test suite size at the intervention. If a quality assurance badge was adopted within 15 days of the intervention, we see a larger effect than for information badges.



F SIGNALS OF BETTER CONTRIBUTIONS

Table 6: Proportion of pull requests containing tests (1-2) per month at time of study (3) monthly

	<i>Dependent variable:</i>		
	proportion		
	<i>logistic</i>	<i>generalized linear mixed-effects</i>	
	Base Model	Full Model	RDD
	(1)	(2)	(3)
log(contributors + 1)	0.235***	0.229***	0.048
log(revisions + 1)	0.160***	0.159***	0.015
log(stars + 1)	0.073***	0.078***	0.048
log(downloads + 1)			0.086***
time_after			0.005
intervention1			0.282*
hasCovBadge1		0.067**	2.307***
hasCIBadge1		0.216***	0.849***
hasOtherBadge		-0.179***	
time_after:hasCovBadge1			-0.037**
time_after:hasCIBadge1			-0.035***
hasCovBadge1:hasCIBadge1		0.155***	-1.866***
intervention1:hasCovBadge1			-0.038
intervention1:hasCIBadge1			0.012
time_after:hasCovBadge1:hasCIBadge1			0.057***
intervention1:hasCovBadge1:hasCIBadge1			-0.172
Constant	-2.783***	-2.897***	-4.305***
AIC	158565	156445	
Projects	3344	3344	325(x19)
Pseudo- R^2	0.129	0.142	

Note:

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

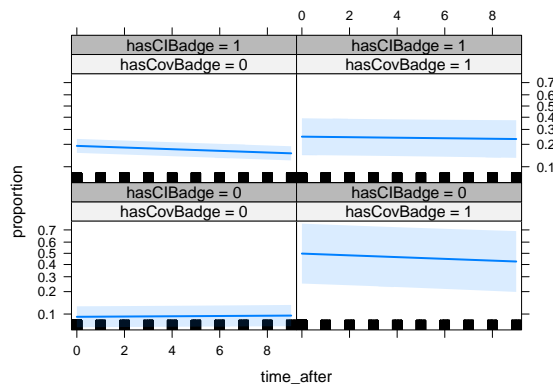
F.1 Effect Plots

We find that the adoption of a quality assurance badge (coverage or continuous integration here) correlates with a small positive change in the proportion of pull requests with tests (second plot below).

Further, projects that adopt both a coverage and continuous integration badge tend to continue to have this increased proportion in the months following the intervention (upper right in the first plot below). All effect plots indicate that the effect decays, but in this case, the decay is slower, or less downward-sloping.

Note that the class (hasCov=1 and hasCI=0) has very few observations.

time_after*hasCovBadge*hasCIBadge effect plot



G SIGNALS OF SUPPORT

Since none of the variables of interest (*intervention* or *time_after_intervention*) have significant effects, we do not provide an effect plot for the longitudinal model.

Table 7: Average issue closing latencies (1-2) per month at time of study (3) monthly

	<i>Dependent variable:</i>		
	log(avgLat + 1)		log(avg + 1)
	<i>OLS</i>		<i>linear</i>
	Base Model	Full Model	<i>mixed-effects</i> RDD
	(1)	(2)	(3)
time			−0.077***
intervention1			−0.043
log(contributors + 1)	0.184***	0.163**	0.304**
log(age + 1)	−0.235**	−0.141	−0.132
log(downloads + 1)	0.028	0.033	0.005
log(revisions + 1)	−0.027	−0.041	−0.194*
log(num + 1)	0.178***	0.178***	
hasSupportBadge1		0.264**	
hasInfoBadge1		0.122	
time_after			0.062
hasSupport1			−0.799***
hasInfo1			−0.108
hasOther		0.054	−0.050
time_after:hasSupport1			−0.089*
time_after:hasInfo1			−0.109*
time_after:hasOther			0.050
Constant	4.292***	3.777***	7.913***
AIC	2192	2191	
Projects	826	826	76(x19)
Pseudo- R^2	0.0492	0.0592	
R_m^2			0.192
R_c^2			0.448

Note:

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

H SIGNALS OF SECURITY

This section was omitted from the paper due to space restrictions. Note that only 15084 projects have a non-zero security score.

For the additional information model, we see that the presence of a dependency management badge increases the chances of having no security vulnerabilities by 59%. Further, if the package has not been updated recently, it is more likely to have vulnerabilities, which agrees with intuition.

H.1 Security Scores

Using the public vulnerability databases of the *Node Security Project* (*nsp*) and *Snyk*, we created a security metric as follows. Each vulnerable dependency gets a score of 1, 5, or 20 depending on whether the database classified it as low, medium, or high priority. Unlike the freshness score, we do not average across all projects. Even a single vulnerability can be a significant fault, whereas out-of-date dependencies do not necessarily cause problems.

Historic security is calculated in the same way as freshness: we only consider those entries in the database happening before the date in history.

Table 8: Security scores (1-2) at the time of study and (3) monthly

	<i>Dependent variable:</i>		
	(security == 0)		log(security + 1)
	<i>logistic</i>		<i>linear</i>
	Base Model	Full Model	<i>mixed-effects</i> RDD
	(1)	(2)	(3)
log(dependencies + 1)	-1.190***	-1.204***	0.019
log(dependents + 1)	0.303***	0.269***	0.062*
log(stars + 1)	-0.113***	-0.122***	0.005
log(contributors + 1)	-0.250***	-0.285***	0.112***
log(tsu + 1)	-0.874***	-0.850***	0.181***
hasExtra		0.288***	
hasDepMgrBadge1		0.290***	
hasInfoBadge1		0.109***	
hasDepMgrBadge1:hasInfoBadge1		0.030	
time			-0.015*
time_after_intervention			0.006
intervention1			-0.030
hasDepmgr1			0.007
hasInfo1			-0.113
intervention1:hasDepmgr1			-0.193
intervention1:hasInfo1			-0.042
hasDepmgr1:hasInfo1			0.087
intervention1:hasDepmgr1:hasInfo1			0.391**
Constant	6.979***	6.853***	0.532**
AIC	91618	91092	
Projects	207854	207854	239(x19)
Pseudo- R^2	0.153	0.158	
R_m^2			0.031
R_c^2			0.236

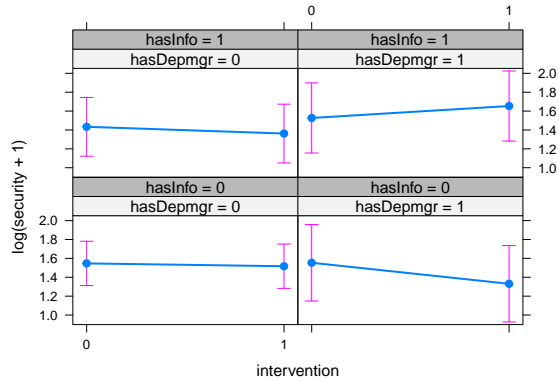
Note:

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

H.2 Effect Plots

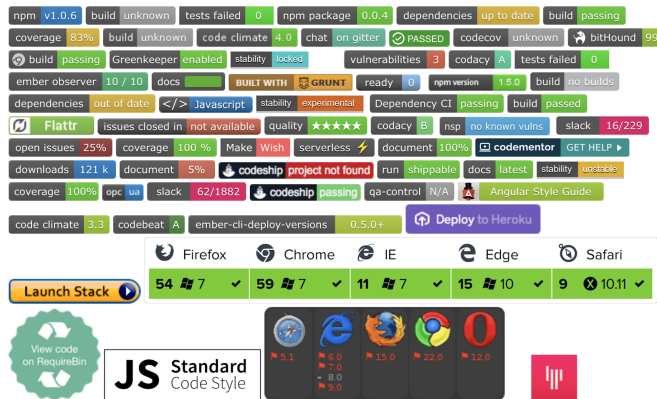
From the significant variable *intervention* : *hasDepmgr* : *hasInfo*, we see that there is a significant positive intervention effect for projects with both dependency management and information badges. This is contrary to our expectations, since a lower security score is better. For other intervention terms, the effect is not significant.

intervention*hasDepmgr*hasInfo effect plot



I SURVEY QUESTIONS

This graphic was displayed at the top of the survey.



I.1 Maintainers

I.1.1 Background Information.

- How many years of experience do you have in open source software?
- How many npm packages are you a maintainer of?
- How many npm packages have you contributed to?
- Which project will your answers below refer to?

I.1.2 Badges in General.

- What influenced your decision to display badges on your project's README file?
 1. They look nice
 2. They display useful information
 3. Advertisements by the badge-offering service itself
 4. Inspired by other projects
 5. Proposed by a collaborator
 6. Common practice in our organization
 7. Other:
- I consider the presence of badges in general to be an indicator of project quality. (five-level Likert scale)

I.1.3 Specific Badges. Here are some badges we found to be popular in the npm community:

- What do you intend to convey to contributors and users through your badges?
 - Please list the most important badges you use and what you are trying to say with them.
 1. Badge 1
 2. Badge 2
 3. Badge 3
- Besides the badges you listed above, are there any other badges you find important? What do they say about your project?
- What effects did you expect each badge would have on your project, contributors, and users?
 - Please refer to the badges that you listed above and explain your answer
 1. Badge 1
 2. Badge 2
 3. Badge 3
 4. Others

I.1.4 Perceived Effects.

- Did you notice any differences in your project's practices, contributors, and users after adding badges? Do you attribute these differences to the badges or to other factors?
- Do you have any additional comments regarding badges?

I.2 Contributors

I.2.1 Background Information.

- How many years of experience do you have in open source software?

- How many npm packages are you a maintainer of?
- How many npm packages have you contributed to?
- Which project will your answers below refer to?

1.2.2 Contributing to Projects.

- When looking for a project to contribute to, I notice badges displayed in the README file. (five-level Likert scale)
- I consider the presence of badges in general to be an indicator of project quality. (five-level Likert scale)
- The presence of badges influenced my decision to contribute to this project. (five-level Likert scale)
- Please explain your previous answer:

1.2.3 Specific Badges. Here are some badges we found to be popular in the npm community:

- What do the badges that you find important tell you about the project?
 - Please name the most important badges that you look for and explain why.
 1. Badge 1
 2. Badge 2
 3. Badge 3
- Are there other badges that you did not list above that you look for? What do they tell you about the project?

1.2.4 Perceived Effects.

- Do you tend to notice a difference in a project's practices or quality depending on the presence of badges? Does this apply to specific types of badges?