Noname manuscript No.
(will be inserted by the editor)

Whom Are You Going to Call?

Determinants of @-Mentions in GitHub Discussions

David Kavaler · Premkumar Devanbu · Vladimir Filkov

the date of receipt and acceptance should be inserted later

Abstract Open Source Software (OSS) project success relies on crowd contributions. When an issue arises in pull-request based systems, @-mentions are used to call on people to task; previous studies have shown that @-mentions in discussions are associated with faster issue resolution. In most projects there may be many developers who could technically handle a variety of tasks. But OSS supports dynamic teams distributed across a wide variety of social and geographic backgrounds, as well as levels of involvement. It is, then, important to know whom to call on, *i.e.*, who can be relied or trusted with important task-related duties, and why.

In this paper, we sought to understand which observable socio-technical attributes of developers can be used to build good models of them being future @-mentioned in GitHub issues and pull request discussions. We built overall and project-specific predictive models of future @-mentions, in order to capture the determinants of @-mentions in each of two hundred GitHub projects, and to understand if and how those determinants differ between projects. We found that visibility, expertise, and productivity are associated with an increase in @-mentions, while responsiveness is not, in the presence of a number of control variables. Also, we find that though project-specific differences exist, the overall model can be used for cross-project prediction, indicating its GitHub-wide utility.

D. Kavaler
University of California, Davis
E-mail: dmkavaler@ucdavis.edu

P. Devanbu
University of California, Davis
E-mail: ptdevanbu@ucdavis.edu

V. Filkov
University of California, Davis
E-mail: vfilkov@ucdavis.edu

1 Introduction

In modern, social-coding [20] projects based on sites like GitHub and BitBucket, that favor the pull-request model, the emergence and growth of a particular type of socio-technical link, @-mentions, can be observed in task-oriented technical discussions. For example, in the *rails* project on GitHub (issue 31804), one of the head developers calls on another, explicitly stating trust of their expertise, saying: *"@kamipo can you take a look since you are our MySQL expert?"* On GitHub, the @-mention in issue discussions is a type of directed social link; the @-mentioner "calls" the @-mentionee via a directed communication that is sent to the @-mentionee through GitHub's interface. Thus, one can consider the network of @-mentions, specifically *calls*, as a sort of directed social network, with a task-oriented purpose. These mentions are heavily used in social coding; in our data, a sample of the most followed and starred projects on GitHub, 52.46% of issues and 22.02% of pull requests contain at least one @-mention, with an average of 1.46 and 1.37 @-mentions per issue or pull request (respectively). On average, developers who are called (while not yet actively participating in the thread) respond 19% of the time; the number rises to *42.94%* when excluding those who never respond[1]. @-mention ubiquity reflects the central role they play in task-oriented social interactions on GitHub. Since much of a developers behavior in OSS projects is recorded, if a person has the expertise and/or are reliable in many different tasks, they will be visible to others. The decision to @-mention someone will be based on visible attributes of that developer, including reliability, productivity, *etc.* Identifying a reliable and knowledgeable person to ask for help or action is key to addressing issues in a timely manner and keeping a project vibrant and alive. In fact, Yu *et al.* found that having @-mentions in a discussion decreases the time to resolve an issue [62]; Zhang *et al.* found that more difficult issues (*e.g.*, longer length of discussion) have more @-mentions [65]. Given these important outcomes, it would be beneficial to know what (observable) socio-technical attributes of developers contribute to being @-mentioned.

As @-mentions have an inherent social element, a global model describing the determinants of @-mention calls would suggest that project-specific social idiosyncrasies are less important than social elements common across GitHub. A global GitHub model for @-mentions may be seen as constructive, since shared social norms across the ecosystem can increase social mobility [54]; on GitHub, this may make the acculturation process easier for those who move between projects. In addition, the findings of Burke *et al.* [11] suggest that those who perceive themselves as socially central contribute more as a result - this may extend to code contributions on GitHub. The findings of Kavaler *et al.* [36] suggest global and project-specific social phenomena (apropos language use) exist on GitHub; is this the case for @-mentions? Or does one phenomenon dominate?

[1] *E.g.*, developers of upstream libraries rarely respond in the downstream project.

The goal of this paper is to understand both the elements contributing to @-mentions in GitHub projects and the extent to which those elements are shared between projects across GitHub. @-mentioning is a complex, multidimensional phenomenon. Developers that are often @-mentioned can have outsized roles and responsibilities in the project network, and be able to handle any task. Thus, a frequently @-mentioned person could well be a strong, highly visible contributor, who might be a trustworthy collaborator on an active task. Whereas visibility can be operationalized more directly, based on a person's aggregate presence in all aspects of the social coding process, both reliability and trust are more complex: we describe the theoretical background for these in the next section. Starting from those theories, and from data on @-mentions and comprehensive developer and project metrics from 200 GitHub projects, we seek a predictive, quantitative model of *future* @-mentions of a developer, using *past* observations of the developer's visibility, expertise, productivity, and responsiveness in their projects. From our quantitative models, together with case studies aimed towards triangulating the model results,

– We find that we can mine a reliable @-mention signal from GitHub data, in ways consistent with current theories in sociology, psychology, and management.
– We see a net positive effect of visibility on @-mentions. We see that less expertise (via, *e.g.*, commits that need fixing, likely buggy commits) associates with lower @-mentions when one has already been @-mentioned, and higher @-mentions if one has not already been @-mentioned; perhaps explained by the idea that any contributions, even defective ones, associates with an initial @-mention, consequently adjusted. We see positive effects for productivity, and none for responsiveness.
– We find that cross-project model fits are generally good, suggesting a common model of @-mentions across GitHub. Similarities among the models are greater for enhanced @-mentions after the first @-mention, than for the initial one.
– We see indications of project-specific @-mentioning behavior, however, the high performance of cross-project prediction suggests the differences may matter little, especially for predicting @-mentions.

We present the theory and research questions in Section 2, research questions in Section 3, data and methods in Section 4, results and discussion in Section 5, practical implications for practitioners in Section 6, threats to validity in Section 7, and conclusions in Section 8.

2 Theory and Related Work

To understand the notion of @-mentions in OSS projects, we build a theory drawing from diverse sources. First, we discuss @-mentions and their use on GitHub, supported by prior work. Then, we introduce theory behind GitHub @-mentioning drawn from work regarding reliability and trust in the fields of sociology, psychology, and management. We then discuss the importance of social exchange and interaction (and thus, the importance of @-mentions) on OSS project success. Finally, we compare our work to that in the field of expertise recommendation.

2.1 @-Mentions on GitHub

GitHub projects have issue trackers with a rich feature set, including ticket labeling, milestone tracking, and code tagging. In GitHub projects, individuals can open up an issue thread where others can comment and discuss a specific issue. In these discussions, developers can tag others using *@-mentions*; the mentioned developer receives a notification that they are being referenced in a discussion. When one decides to @-mention another developer, there is generally a specific reason, *e.g.*, to *reply* to a single person in a discussion involving many others; or, to *call* the attention of someone who isn't currently in the discussion. The latter aspect is what we wish to capture; calling upon another person is an implicit (and on GitHub, often explicit) statement of belief that the receiver could help address the task at hand. To validate the importance of modeling *call* @-mentions on GitHub, we perform a case study (Section 5.1) and also look to prior literature (below) for the reasons behind the use of @-mention.

Tsay *et al.* performed interviews with several developers of popular projects on GitHub, specifically related to the discussion and evaluation of contributions [59]. They found that both general submitters and core members use @-mentions to alert core developers to evaluate a given contribution or start the code review process. They further found that core members often @-mentioned other core members specifically citing that the @-mentionee is more qualified to answer a particular question or review a given contribution. In nearly all cases, the @-mention seems to be used to draw the attention of a developer who may contribute to the task at hand. Kalliamvakou *et al.* surveyed and interviewed developers, mostly commercial, that use GitHub for development [35]. Of all interviewees, 54% stated that their first line of communication is through the @-mention[2]. In addition, they state that teams often use the @-mention to draw members' attention to a problem. [3]

[2] Developers were asked about communication methods, not explicitly the @-mention.

[3] Described in Section 4.2, a reply @-mention is directed towards someone already in the discussion; a call @-mention is directed towards someone not yet in the discussion. In our data, there is indeed a very high correlation between reply @-mentions and discussion length (0.812); however, there is a relatively low correlation between call @-mentions and discussion length (0.283). As our focus is on call @-mentions, correlation between reply @-mentions and discussion length is not a threat.

2.2 @-Mentions and Personal Reliability

The ability to rely on others socio-technically is critical for cohesive workgroups. From a social perspective, Saavedra *et al.* argue that reliable interactions among group members are important for success, especially when tasks are interdependent [53]. According to social learning theory, frequent interactions among group members increases the likelihood that some in the group will be raised to "role model" status [3,4]. The importance of role models in social learning has been widely discussed [4,11,21]. On GitHub, researchers have found that these role models ("rockstars") are important influencers, allowing developers to learn from "rockstar" code contributions in order to improve their own work [20,39]. In other words, developers rely on others within and outside their immediate working group in order to solve problems. In addition, peer code review (relying on team members other than the authors for manual inspection of source code) is recognized as a valuable tool in software projects [1]. Thus, we argue that identifying these reliable developers, by means of the @-mention, is important for project success. We theorize that reliability will manifest itself on GitHub through *responsiveness*, measuring: if you are called, how often do you answer?

2.3 @-Mentions and Trust

Trust has a long-recognized complex [42,24] social component and well understood benefits to social and economical well-being [32,45], in both physical and virtual teams [33]. While individuals do have a personal notion of when to trust someone, in social settings those notions inherit from the communal sense of trust [45,33,32]. In socio-technical groups like software projects, contributors must be trusted as technically competent, and also as useful to the project. Gaining contributor status is a key indicator of trust, which has been extensively studied [6,56,15,25,22]. In pull-request oriented models, with *decentralized* repositories, anyone can make changes in a fork, and then submit the changes as a pull-request. Here, social processes such as code-review take a central role in deciding the fate of code contributions. Opinions from trusted people during the relevant discussions would be in great demand, and thus, the social demand on a person is an indication of the trust placed upon them by the community. Since the pull-request model is more or less normative in GitHub projects, it is reasonable to posit that many projects in the GitHub community ecosystem may share the same determinants @-mention extension, *i.e.*, the reasons behind @-mention extension may be a global phenomenon.

We acknowledge that an @-mention does not necessarily arise purely to trust in the taggee; however, some form of trust likely plays a role. Thus, understanding theories of trust is important to understanding @-mentions on GitHub.

Oft-mentioned and widely discussed, the meaning and role of trust has been examined across many disciplines, including sociology, psychology, and philosophy [66,10,38,9,30]. Gallivan provides a succinct set of definitions for trust types as provided by prior work on organizational trust [24]; relevant types for GitHub are: 1) *Knowledge-based trust*: trust based upon a prior history of transactions between two parties; 2) *Characteristic-based trust*: trust that is assumed, based on certain attributes of the other party; and 3) *Swift trust*: a "fragile" form of trust that emerges quickly in virtual workgroups and teams.

For our work, the idea of swift trust is important as it is theoretically defined for virtual teams, as on GitHub. Jones and Bowie [34] state: "*the efficiency of [virtual teams] depends on features - speed and flexibility - that require high levels of mutual trust and cooperation*"; other researchers share and expand on this notion [47,28]. Though swift trust may initially appear most applicable, much of the founding work was done prior to the proliferation of socio-technical systems such as GitHub. More recently, Robert *et al.* redefine swift trust for modern systems as a combination of classical swift trust, knowledge-based trust, and parts of characteristic-based trust [50]. We agree with this blended definition - a sweeping categorization of GitHub as having a swift trust system is likely incomplete; multiple trust regimes probably apply. We capture knowledge-based trust through our measures of *visibility*, *i.e.*, functions of @-mention network degree. Characteristic-based trust is also likely; task characteristics can be easily seen on GitHub, as captured by measures of *expertise* and *productivity*.

2.4 @-Mentions and Social Exchange

On GitHub, the @-mention is a type of directed social link; the @-mentioner causes a notification to be sent to the @-mentionee through GitHub's interface, a form of social communication. Thus, the network of @-mentions is a sort of social network, with a task-oriented purpose. Much work has been done in variety of fields on identifying reasons behind social tagging and mentioning behavior, including on GitHub [63].

In the fields of psychology and sociology, many researchers have explored the phenomenon of social tagging on Facebook [49,12,46]. In general, this research has shown that social tagging provides a sense of community and increases one's social capital. These findings are of importance to GitHub as they elucidate the importance of community social interaction, which are known to be important to OSS success [26,25]. Of specific interest, Burke *et al.* found that those who receive feedback on their Facebook posts share more [11]. It is reasonable to believe that this extends to task-oriented networks, such as GitHub; those who feel as though their contributions are important, socially or technically, are likely to contribute more.

McDonald *et al.* interviewed multiple GitHub developers and found that they rarely use product-related measures (*e.g.*, release quality, bug fixes) to describe project success; rather, they use measures such as number of (new) contributors, pull requests, *etc.* [41]. As stated above, social exchange is important to both one's own well-being and OSS success. As social measures have been shown to be important for OSS *product* success [29], and given that developers generally use non-product measures to describe *project* success, fostering the use of @-mentions and thus the exchange and gain of social capital would be beneficial for both metrics of success. We capture social aspects in *visibility* - functions of @-mention network measures.

2.5 @-Mentions and Discourse/Dialogue

Discourse and dialogue have seen a resurgence of research interest with the advent of NLP computational methods. Stolcke *et al.* [58] have most prominently defined discrete conversational speech categories into which @-mentions fit well, perhaps because they themselves are social link extensions. Stolcke's *et al.* [58] work and the other aforementioned prior work [59,35], helped us distill the following four categories of speech that use @-mentions (one of these is a slightly modified category as compared to Stolcke's work, marked by ⋆):

1. **Request (R)**: An explicit request towards the called person to perform some action.
2. **Request-Suggest (R-S)**: An implicit request towards the called person to perform some action.
3. **Inform (I)**: An indication that the issue or post is relevant to the called person.
4. ⋆**Credit Attribution (CA)**: An @-mention designed to attribute credit to the called person. This is similar to "Thank" by Stolcke *et al.* [58], but explicitly directed at an individual.

We use these categories in a case study examining reasons behind call @-mentions in Section 5.1.

2.6 Expertise Recommendation

As our interest is in @-mentions, often used to call upon those with expertise relating to the task at hand, we compare our work with that in the field of expertise recommendation. Cubranic and Murphy [44] used text classification on data from the Eclipse bug tracking system in order to identify developers relevant for a given bug; Matter *et al.* performed a similar study [40]. Likewise, Mockus and Herbsleb developed and deployed a tool to identify expert developers [43] working on proprietary software. More directly relevant to our work, Ibrahim *et al.* developed a tool to identify which developers should contribute to a given discussion on mailing lists for three open source projects [31]. Though our work is similar in spirit to that in the field of expertise recommendation, our focus is specifically on the calling signal itself (@-mention). This goal is unique from expertise identification and recommendation as our goal is to identify reasons behind the signal extension itself, rather than identifying the best receiver of said signal.

3 Research Questions

@-mentions signal a desire for a developer's involvement in a task-oriented discussion. GitHub is a rich source of mine-able, potentially relevant, developer characteristics.

The theory above allowed us to identify relevant dimensions along which to model the phenomenon of @-mentions. We describe them shortly here, and operationalize them in the Methods section. *Visibility* measures the ability of others to know of a developer; if a developer is to be @-mentioned, people must know the network in order to know who they are capable of reaching. *Expertise* can be defined through task-related measures, *e.g.*, number of likely buggy commits, which might influence how much a developer is @-mentioned. *Productivity* is defined by number of commits; prolific committers could be viewed as the "top brass" of a project, and commits are easy to see in GitHub. Finally, we are interested in *responsiveness*; if a mentionee is called to lend their talent, it is not farfetched that those who respond to the call are more likely to be @-mentioned in the future.

We explicitly model *future* @-mentions, *i.e.*, @-mentions as measured 6 months beyond the "observation period", described further in Section 4.6. Having an effective model that explicitly predicts future behavior has higher utility to potential future applications than an aggregate regression model over the whole history.

RQ 1: Can we describe/predict future @-mentions in terms of developer visibility, expertise, productivity, and responsiveness?

Our second question relates to the utility of our model. If one wishes to use our model on their own projects, it would be helpful to be able to use the model pre-trained on some data, *e.g.*, trained entirely on a separate project and applied to one's own.

RQ 2: Can models trained entirely on one project be reliably used to predict @-mentions on another project?

Our third question is more theoretical in nature. Specifically, we wish to describe the differences between projects in terms of our determinants of @-mentions and identify some potential reasons behind these differences. As GitHub is composed of subcommunities which may have some idiosyncrasies, we believe that these differences may be reflected in our describers of @-mentions.

RQ 3: Is there evidence of project-specific @-mention culture? Or are the determinants of @-mentions a GitHub-wide phenomenon?

4 Data and Methodology

All data was collected by querying GitHub's public API using the Python package PyGithub[4], with the exception of issue fixing data, which was gathered by cloning individual repositories. Commits are found through the official

[4] `https://github.com/PyGithub/PyGithub`

GitHub API, including commits within and without pull-requests. Developers for a given commit are identified automatically by inspecting the commit's specified *author* within git, and querying GitHub for an existing user with the same name. If no GitHub user is found, the commit is not attributed to any user in our data. Data was gathered during the month of July 2017.

4.1 Filtering and Cleaning

For our data set, we looked at the top 900 most starred and followed projects, each of which likely to contain enough issues and commits for us to model robustly. The number of stars and followers are proxies for project popularity. We noted that among the 900 projects there was a significant difference in popularity among the 1st and the 900th project. Some of the measures we used are expensive to calculate, so we had to limit our calculations to a smaller sample. We chose as our sample a random subset of 200 of those 900 projects. The choice of a random sample, as opposed to, say, the very top 200, was made to ensure we captured a diverse mixture of projects in terms of popularity, and, thus, have a more widely applicable results from the modeling.

Due to our described project selection method, it is possible that non-software projects are within our final sample. However, our research questions are not software specific; our findings are meant to reflect the whole of GitHub - primarily software, but also projects focused on other goals (*e.g.*, books and link collections).

We ran multiple parallel crawlers on these 200 projects to gather commits, issues, pull requests, and associated metadata. Due to some internal issues with the PyGithub package[5], some projects failed to return complete data. We created a verification system (completely external to PyGithub) to determine which projects were incomplete, and removed them from consideration. Finally, we only consider developers with at least one commit to their given project in order to avoid a proliferation of zeros in our covariates, as many developers participate in issue discussions but never contribute. This was done in order to focus on those who may become @-mentioned in the future; without any commits, we argue it is unlikely to be @-mentioned in the future. To support this claim, we note that in our data, the average number of future @-mentions for those with zero commits is 0.382 with a standard deviation of 1.98 commits; in contrast, the average number of future @-mentions for those with more than zero commits is 1.98 with a standard deviation of 10.62.

As we wish to explicitly model future @-mentions, we introduce a time split in our data. For each project, we define a time frame under which we "observe" the project and its participants, and model our response as calculated beyond our observation time frame - the "response" period. We decided to set our response period as 6 months, *i.e.*, 6 months prior and up to the end of our data. We also tested periods of 3 and 12 months; 3 months had little difference to 6

[5] PyGithub did not handle properly some Null responses from GitHub's API.

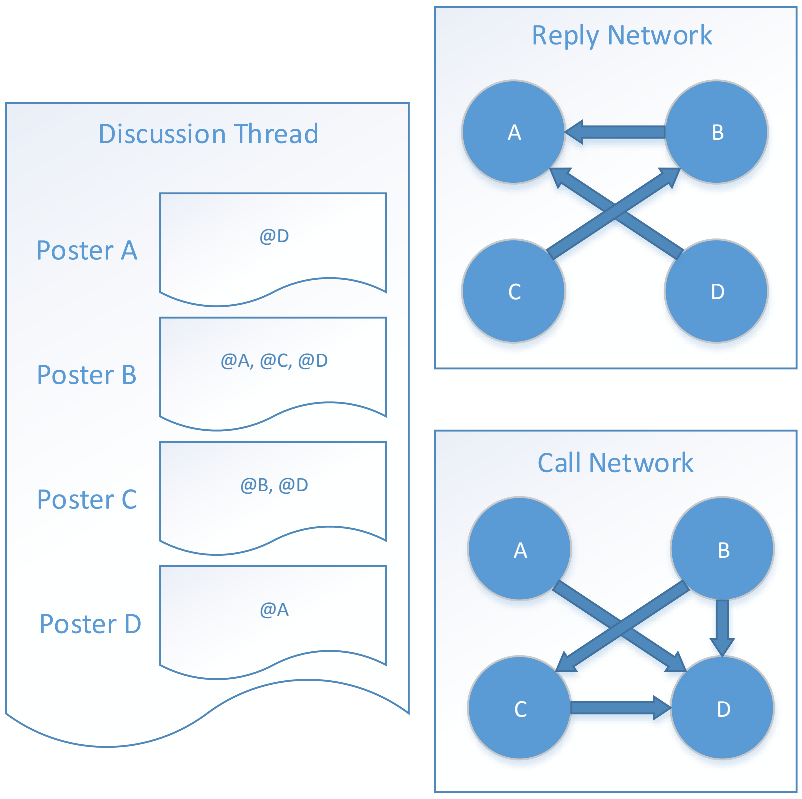


Fig. 1: The network creation process. Shown is a discussion thread and the resulting reply and call networks. Note this can be a multigraph (not shown).

months, and 12 months left us with too little data to model. We then filtered out each individual who had a project participation shorter than 3 months. This is because with 2 months of data, there exists only one line which can be fit: the line with the first month as the start point, and second month as the end point. With 3 (or more) data points, there exists more between- and within-subject variance to be captured by the model, further reducing the worry of overfitting, as there are multiple lines which may fit ≥ 3 data points. Thus, we explicitly model future @-mentions, as our response period is disjoint from our observation period. In total, this yielded 154 unique projects comprised of $17,171$ project-developer pairs to test our hypotheses.

4.2 Issues and @-Mentions

For each project on GitHub, individuals can open up an issue thread where others can comment and discuss a specific issue[6]. We constructed a social network for each project using @-mentions in their issue comment threads; Fig. 1 depicts this process. Similar to Zhang *et al.* [64], *i.e.*, every edge (u, v) is developer u @-mentioning v somewhere in their post. This yields a directed multigraph; there can be multiple edges (u, v) depending on how many times u @-mentions v. We distinguish between two edge types: *reply* and *call*. A *reply* edge is defined by u @-mentioning v when v *has already posted in the given thread.* A *call* edge is defined by u @-mentioning v when v *has not yet posted in the given thread.* Thus, a call edge is representative of the phenomena we wish to measure, described in Section 2; u calls upon v as u wishes for v's input for the discussion at hand.

4.3 Focus

As a measure of visibility, we wished to capture phenomena more nuanced than merely raw indegree and outdegree[7], as raw degree counts do not take into consideration the larger, neighborhood view. Standard global measures used in social network analysis are often too expensive to calculate for our large @-mention networks[8]. Thus, we require a measure that takes into account a more global view that is relatively inexpensive to calculate. Here, we introduce the idea of *social focus* in the @-mention network.

Theoretically, we believe that when given many choices on who should be contacted (@-mention), individuals must make a decision, based on their knowledge of the potential receiver's characteristics (*e.g.*, ability to help in a task) and who is more readily visible. In social networks, knowledge of others is propagated through existing links. Thus, if an individual is highly focused-on, it is likely that they will become more so in the future. This means that the more focused-on a developer is, the more visible they likely are. In addition, those who have lower social focus on others, *i.e.*, they distribute their out-links widely among many others, are also more likely to be visible to others.

To represent focus, we adapt a metric described by Posnett *et al.* [48]. This metric is based on work by theoretical ecologists, who have long used Shannon's entropy to measure diversity - and its dual, specialization - within a species [27], and can be derived from Kullback-Leibler divergence. For discrete probability distributions P and Q, Kullback-Leibler divergence (KL) is defined as:

$$D_{KL}(P|Q) = \sum_i P_i \ln \frac{P_i}{Q_i}$$

[6] Note that pull requests are a subset of issues.

[7] Though we do use outdegree in our model as well.

[8] *E.g.*, standard algorithms require a full adjacency matrix to be in memory at once; memory will be exhausted for networks of our size.

Bluthgen *et al.* define a species diversity measure, δ[9], using D_{KL} [8]. This measure is calculated naturally in a bipartite graph formulation, where each species in the graph has its own diversity value δ_i. Posnett *et al.* use this metric, normalized by the theoretical maximum and minimum (*i.e.*, so δ_i ranges from 0 to 1), to measure "developer attention focus" (DAF) [48]. When δ_i (a row-wise measure) is high, developer i is more focused in commits to a fewer number of modules. Analogously, when δ_j (a column-wise measure) is high, module j receives more focused attention from fewer developers. They call these quantities "developer attention focus" ($\mathcal{DAF}_i$) and "module attention focus" ($\mathcal{MAF}_j$)[10].

In this work, we take these definitions and expand them to the social network of @-mentions. Recall that we distinguish between two types of @-mentions: *reply* and *call*. We can likewise represent our social network as a bipartite graph, where the rows and columns of the adjacency matrix both refer to developers, and each cell s_{uv} is the count of directed @-mentions from developer u to developer v for a given @-mention type. Thus, we analogously define ρ_u as the focus developer u gives in their reply @-mentions, and ρ_v as the focus developer v receives from others' reply @-mentions. Similarly, we define κ_u as developer u's focus in their call @-mentions, and κ_v as the focus developer v receives from others' call @-mentions.

Recall that we can interpret these values equivalently as a measure of *specialization* or *inverse uniformity*. For example, if ρ_u is large, developer u specializes their replies to a select group of others; if ρ_u is small, developer u uniformly replies to all others. Likewise, if κ_v is large, developer v is called by a select group of others; if κ_v is small, developer v is called uniformly by all others. We believe this intuition is useful to answer our research questions. Thus, we define normalized *outward social specialization* and *inward social specialization* measures for both replies (ρ) and calls (κ):

$$\mathcal{OSS}_{u,\rho} = \frac{\rho_u - \rho_{u,min}}{\rho_{u,max} - \rho_{u,min}} \qquad \mathcal{ISS}_{v,\rho} = \frac{\rho_v - \rho_{v,min}}{\rho_{v,max} - \rho_{v,min}}$$

where $\mathcal{OSS}_{u,\kappa}$ and $\mathcal{ISS}_{v,\kappa}$ are defined analogously.

4.4 Attributing Commits That Need Changing

To identify commits that had to be changed in order to close an issue (*i.e.*, likely buggy commits), we use the standard SZZ algorithm [55], as expanded in [37], with a few changes to accommodate GitHub nuances. GitHub has a built-in issue tracking system; developers close open issues by using a set of keywords[11] in either the body of their pull request or commit message. *E.g.*, if a developer creates a fix which addresses issue #123, they can submit a pull request containing the phrase "closes #123"; when the corresponding

[9] This measure is originally called d by Bluthgen *et al.*, but we will use δ here to reserve d to represent developers.

[10] We do not use $\mathcal{MAF}$, we use an analogous form for our social networks.

[11] `https://help.github.com/articles/closing-issues-using-keywords/`

fixing patch is merged into the repository, issue #123 is closed automatically. To identify likely bug-fixing commits, we search for associated issue-closing keywords in all pull requests and commits. We then "git blame" the respective fixing lines to identify the last commit(s) that changed the fixing lines, *i.e*, the likely buggy lines. We assume the latest change to the fixing lines were those that induced the issue, and refer to those changes as likely buggy, or buggy for short.

We note that an issue is a rather broad definition of a bug, as an issue can be brought up to, *e.g.*, change the color of text in a system's GUI; this may not be considered a bug by some definitions. However, as GitHub has the aforementioned automatic closing system, we believe that our identification of fixing commits (and therefore buggy commits) does not contain many false positives. Prior work has relied on commit message keyword search, which may introduce false positives due to project-level differences in commit message standards, *i.e.*, what a commit message is expected to convey. These standards can vary widely [7].

4.5 Variables of Interest

We are interested in measuring and predicting @-mentions as a function of *readily observable* developer attributes, namely *visibility*, *expertise*, *productivity*, and *responsiveness*. We operationalize these attributes as follows:

We define **visibility** as the ability for developers to note a person's existence; if developer A is not aware of the existence of developer B, it is unlikely that A would @-mention B. This is akin knowledge-based trust. Here, we use our social specialization measures $\underline{\mathcal{OSS}_\rho}$, $\underline{\mathcal{OSS}_\kappa}$, and $\underline{\mathcal{ISS}_\kappa}$, along with total social outdegree (total number of @-mentions for a developer in a given project) as measures of visibility. We believe these measures are reasonable as they identify one's existence within the social network.

We define **expertise** as a developer's ability to complete project tasks in accordance with team expectations, related to characteristic-based trust. To represent this, we use number of issue-inducing commits made by a developer, focus measure $\mathcal{DAF}$, and a factor identifying whether or not the given developer is the top committer or project owner. A higher number of issues can indicate a lack of aptitude for programming according to the project's goals[12]. It has been shown that a higher $\mathcal{DAF}$(*i.e.*, higher module specialization) is associated with fewer bugs in a developer's code [48]. Thus, $\mathcal{DAF}$ can represent developers' expertise in code modules. The top committer or project owner factor indicates a certain level of prestige and expertise; one would expect the top contributor or project owner would likely be the most expert in matters concerning the project. Number of fixing commits was also calculated, but was not used due to collinearity with that of bug commits.

[12] We use issues fixed before closing as proxy for bugs; a higher value need not imply lack of aptitude, but it indicates a change in expected coding behavior and expertise.

We measure **productivity** as the raw commit (authoring) count. Measures of productivity abound– most have been shown (of those we computed, *e.g.*, lines of code added or deleted) to highly correlate with commit count, especially in models where confounds are recognized. We choose commit count as it is the simplest.

We describe **responsiveness** as a measure to answer the question: when you are called, do you show up? One would expect that those who are responsive, and thus display their reliability, will be called upon again. This is precisely defined as the number of times a developer is called and responds to that call; *e.g.*, if a developer is called in 10 unique issues and responds in 8 of those issues, their responsiveness value is 8.

4.5.1 Extra-Project Controls

As stated, our interest is to identify *readily observable* attributes of potential @-mentionees (*e.g.*, within-project social activity and commit activity), and functions thereof. This is in contrast to things that may be hard to observe, such as activity outside the project at hand (*e.g.*, outside-project social activity, exact number of commits to other projects, *etc.*). However, such a control for outside experience is likely necessary as, *e.g.*, a developer that is experienced outside the project may already be known due to outside channels, and thus have an inflated likelihood of being @-mentioned to begin with. We consider an outside-project attribute, developer's GitHub age (in days), in order to control for experience outside the project which may lead to increased @-mentions when project contributions are relatively low. As GitHub age is readily observable through the profile interface on GitHub (*e.g.*, by viewing the contribution heatmap), we believe this to be a reasonably observable control. Another outside-project control we considered was number of public repositories contributed to by a developer, as this is readily observable; however, this was highly correlated with age, and was thus dropped from the model.

4.6 Modeling Future @-Mentions

To answer our questions, we use count regression in a predictive model. This allows us to inspect the relationship between our response (*dependent variable*) and our explanatory variables (*predictors* or *covariates*, *e.g.*, responsiveness) under the effects of various *controls* (*e.g.*, project size).

There are many forms of count regression; most popular are so-called Poisson, quasi-poisson, and negative binomial regression, all of which model a count response. In our work, we are interested in @-mentions as measured by number of incoming @-mention calls per person - a count. In addition, as our data contain many zeros, we need a method that can accommodate; the methods listed above all have moderate to severe problems with modeling zeros. *Zero inflated negative binomial regression* and *hurdle regression* are two methods specifically designed to address this challenge by explicitly modeling the existence of excess zeros [14]. It is common to fit both types of models, along with a negative binomial regression, and compare model fits to decide which structure is most appropriate. Standard analysis of model fit for these methods uses both Akaike's Information Criterion (AIC) and Vuong's test of non-nested model fit to determine which model works best [61].

We employ *log* transformations to stabilize coefficient estimation and improve model fit, when appropriate [18]. We remove non-control variables that introduce *multicollinearity* measured by *variance inflation factor (VIF)* > 4 (*e.g.*, we do not use $\mathcal{ISS}_\rho$ due to high VIF), as multicollinearity reduces inferential ability; this is below the generally recommended maximum of 5 to 10 [18]. Keeping control variables with high VIF is acceptable, as collinearity affects standard error estimates; as control variables are not interpreted, we do not much care if their standard error estimates are off [2]. We model on the person-project level, *i.e.*, each observation is a person within a project. We performed multiple hypothesis testing (p-value) correction by the Benjamini and Hochberg method [5]. A squared age term is present in the zero model to account for a quadratic shape in the residuals, along with its lower order term as is standard in regression [23].

As noted in Section 3, we explicitly model future @-mentions; our response variable is the value 6 months after our "observed" (*i.e.*, covariate) data. As such, we build a *predictive* model, not a fully regressive model - *i.e.*, one that is built on the entirety of available data. We note the difference is minor, but worth reiterating.

5 Results and Discussion

5.1 Case Study: Project-Level Reasons for Call @-Mentions

We are interested in empirically measuring the reasons behind the @-mention. To make sure our theoretical underpinnings are reasonable, we performed a random manual inspection of 100 call @-mentions from our data set, to qualitatively identify the primary reason behind the call. A sample of size 100 grants far above the recommended statistical power of 0.8 at an error rate of 5% for 5 pairwise comparisons (for our 5 qualitative groups, discussed below) based on 1-way ANOVA with 2-sided quality; thus, our results are considered statistically sound [16]. Though call @-mention use might be more common in some projects than others, our sample of 100 call @-mentions were selected at random across all projects, and thus our results should not be biased.

This study was performed initially by one author, with qualitative codes defined by prior work [58] (discussed in Section 2.5) and in collaboration with another researcher (not credited as an author of this work). We first identified a set of general codes from the mentioned prior work, and consolidated these into the final set of 5 presented. The goal of this case study was to validate theoretical reasoning discussed in Section 2, specifically to identify whether or not our definition of a call @-mention is viable, as well as guiding decisions regarding variables for our models. Thus, although our observed counts within each category are statistically robust as defined by recommended procedure, this study was performed primarily as a motivational study, rather than a strictly statistically robust undertaking.

The counts of each category found in our manual inspection is shown in Table 1. In the case of **R**, we argue that reliance and/or trust in the mentionee is clear: the mentioner explicitly requests that the mentionee performs some defined task; if the mentionee was deemed unreliable, the mentioner would be unlikely to trust them with an explicit task.

For **R-S**, the mentionee is not explicitly called upon to perform some task. However, the mentioner seems to want the mentionee to respond (or perform a task), but does not wish to explicitly tell the mentionee to act, likely out of politeness. Though the call to action is not explicit, we argue this still represents mentionee reliability; like **R**, the mentioner wants the mentionee to perform an action, but does not explicitly state as much.

In the case of **I**, the call is meant to tag the @-mentionee in case they want to participate; not necessarily in order to respond to the thread, or perform some action. However, the mentioner believes that the mentionee may be interested in the issue at hand. This is similar to **R-S**, albeit slightly weaker, as the mentioner may not have a particular task in mind for the mentionee. However, this still indicates that participation from the mentionee may be appreciated. E.g., in Table 1, in the **I** example, @DavidGoll @karelz are cc-ed in a message to @mconnew, to inform them of a new development in a discussion they were previously involved.

In the case of **CA**, the mentioner is calling the mentionee in order to give credit, *e.g.*, when the mentionee produced an important patch that is relevant to the discussed issue. Though this is not a clear reliance on the mentionee in description, in practice we find it is often used in a similar way to **I**; participation from the mentionee may be appreciated, but not necessary.

Across all 100 manually inspected cases, we found only 3 cases in which the call @-mention does not fall into the aforementioned categories (3%); one appears to be a misuse of the @-mention; the other two are due to users changing their GitHub display name after the @-mention is seen, thus throwing off our detection of the @-mention as a call rather than a reply. Thus, we argue that the call @-mention is consistently representative of reliance on the mentionee.

5.2 Future @-Mention Models

Fig. 2 shows a selection of variables from our categories of interest and their paired relationship with future @-mentions. For all variables, we see a strong positive relationship with @-mentions; the largest correlation sits with developer responsiveness (78.90%).

Table 1: Call @-mention categories, samples, and case study.

Category	Example	Count
R (Request)	*Project: hashicorp/terraform; Issue:* 7886 "@phinze - can we please have someone take a look at this PR now that tests and docs are complete?"	39
R-S (Request-Suggest)	*Project: dotnet/roslyn; Issue:* 18363 ""... I don't know if the test flavor recognizes this capability. @codito @sbaid would know."	17
I (Inform)	*Project: dotnet/corefx; Issue:* 8673 "/cc @DavidGoll @karelz @mconnew My current understanding (based on WinHTTP's response) is ..."	33
CA (Credit Attribution)	*Project: avajs/ava; Issue:* 1400 "... There is already a PR for this though, thanks to @tdeschryver ..."	8
Misuse or Misclassification	*Project: celery/celery; Issue:* 817 "We are also using them in production @veezio for quite some time, works fine." *Author's note: @veezio is a company GitHub account.*	3

Though paired scatter plots provide initial insight to determinants of potential power, we must model them in the presence of other variables, along with controls, to properly answer our questions.

RQ 1: Can we describe/predict future @-mentions in terms of developer visibility, expertise, productivity, and responsiveness?

Table 2 shows our model of future @-mentions, with determinants of interest grouped and separated from one another. Our analysis points to a zero hurdle model as providing the best fit, which separately models the process of attaining one's first call ("zero" model, logistic regression), and the process of attaining beyond one call ("count" model, negative binomial regression). We tested for the usage of a negative binomial regression as opposed to a Poisson regression in the count model by fitting an additional quasi-poisson model to test for issues of under- and over-dispersion, which can be an issue for Poisson models and may affect model quality. Quasi-poisson models explicitly fit a dispersion parameter which can be used to evaluate under- and over-dispersion; negative binomial models can be used for over-dispersed data [51]. Results from these tests suggested that over-dispersion was an issue for our data; in addition, a negative binomial count model had a better fit than both Poisson and quasi-poisson models. Thus, we chose to model non-zeroes using a negative binomial model in our hurdle regressions. Fig. 3 depicts predicted and observed values along with a $y = x$ and trend line.

The mean average and mean squared error are 0.910 and 15.769, respectively. To aid in interpretation of how good this predictive model is, we note that the range for the observed future @-mentions is from 0 to 136, with an average of 2.637; thus our mean average error with respect to the spread is $0.910/173 = 0.670\%$ and with respect to the average is $0.910/2.380 = 38.24\%$. In other words, looking at the mean average error, our model differs from

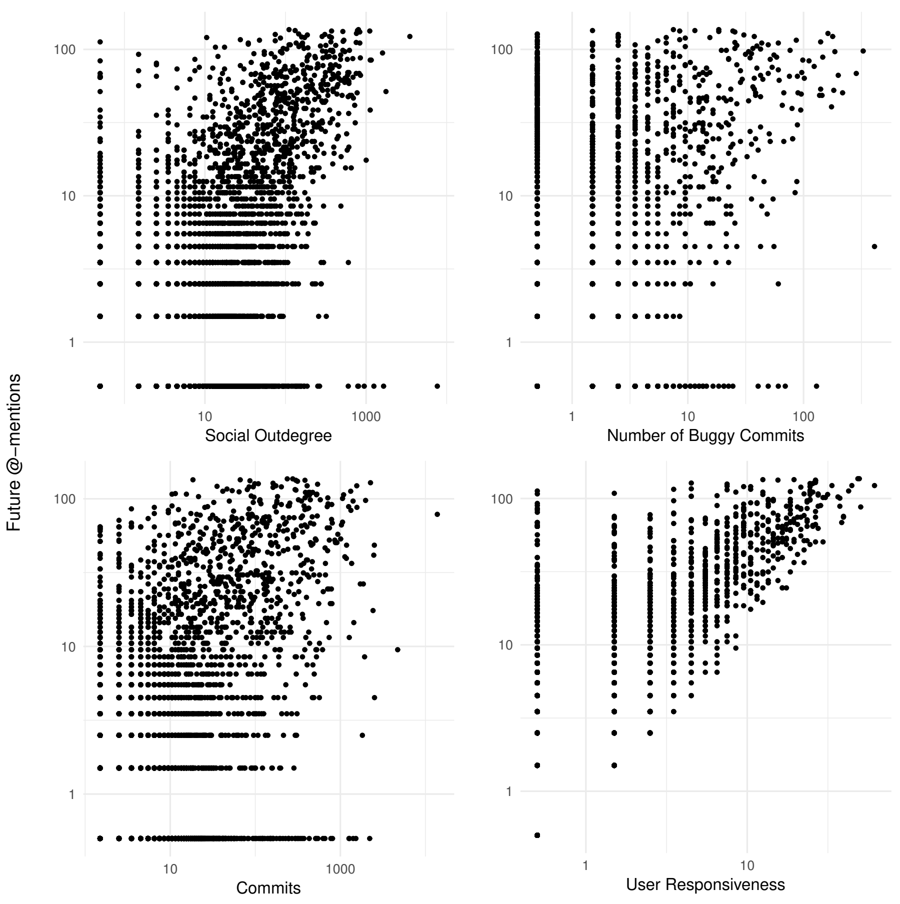


Fig. 2: Future @-mentions vs. selected attributes of visibility, expertise, productivity, and responsiveness. Axes log scaled.

the observed value by an average of 0.910 call @-mentions. Thus, we say this indicates a good model fit.

Visbility We see that $\mathcal{OSS}_\rho$ and social outdegree are positive for both the count and zero components of our model. This suggests that a higher social focus (in replying to others) and larger overall social outdegree associates with being @-mentioned in the future - be it in the transition from zero to greater than zero @-mentions, or in increasing @-mentions. However, we see a negative coefficient for $\mathcal{ISS}_\kappa$, suggesting that when others focus their calls on the observed individual, the observed's @-mention count decreases[13]. This negative coefficient is not unexpected; $\mathcal{ISS}_\kappa$ is derived from the Kullback-Leibler divergence, and when there are many cells (*i.e.*, others that can be called), it is expected that a higher focus is correlated with a lower raw value.

[13] $\mathcal{ISS}_\kappa$ is not used for the zero component; it is undefined when call mentions are 0.

Table 2: Future @-mention model; p-values corrected by BH method. User subscripts omitted; they refer to the developer under observation within the model

	Dependent variable:			
	Future @-mentions (6 months later)			
	Count	(Std. Err.)	Zero	(Std. Err.)
Visibility				
$\mathcal{OSS}_\rho$	0.103*	(0.045)	0.351***	(0.100)
$\mathcal{OSS}_\kappa$	−0.046	(0.040)	0.508***	(0.099)
$\mathcal{ISS}_\kappa$	−0.283***	(0.047)		
Log Social Outdegree	0.058***	(0.008)	0.433***	(0.022)
Expertise				
Log Number of Buggy Commits	−0.065***	(0.010)	0.187***	(0.043)
$\mathcal{DAF}$	−0.040	(0.042)	−0.134	(0.101)
Top Committer or Project Owner	0.055	(0.044)	0.691	(0.534)
Productivity				
Log Commits	0.086***	(0.008)	0.453***	(0.025)
Responsiveness				
Log User Responsiveness	−0.003	(0.012)		
Controls				
Committer Only	0.141***	(0.039)	−1.584***	(0.060)
Log Total Posts in Project	0.021*	(0.010)	0.151***	(0.021)
Log Observed @-Mention Value	0.980***	(0.011)		
User GitHub Age (Days)	−0.137***	(0.020)	−1.470***	(0.430)
User GitHub Age (Days) Squared			0.116***	(0.031)
Intercept	0.637***	(0.180)	1.684	(1.511)
Observations	17,171			
Mean Absolute Error	0.910			
Mean Squared Error	15.769			

†p<0.1; *p<0.05; **p<0.01; ***p<0.001

E.g., consider the case where 10 individuals can call on developer A. If each calls A once, the raw value for calls is 10 and $\mathcal{ISS}_\kappa$ is low; if only one developer calls A, the raw value is 1 but $\mathcal{ISS}_\kappa$ is high. In support of this intuition, Posnett *et al.* [48] found that a higher value of $\mathcal{DAF}$ associates with a lower raw cell count.

In sum, having a larger social presence ($\mathcal{OSS}_\rho$, social outdegree) may associate with one's future @-mention count. These values are much easier to increase for an individual than $\mathcal{ISS}_\kappa$, as $\mathcal{ISS}_\kappa$ is a function of indegree, and thus less in the individual's control.

Expertise The number of likely buggy commits a developer makes has a negative coefficient for the count component, suggesting that a larger number of likely buggy commits associates with a decrease in @-mentions. This is as expected: a higher expertise should lead to more future @-mentions. However,

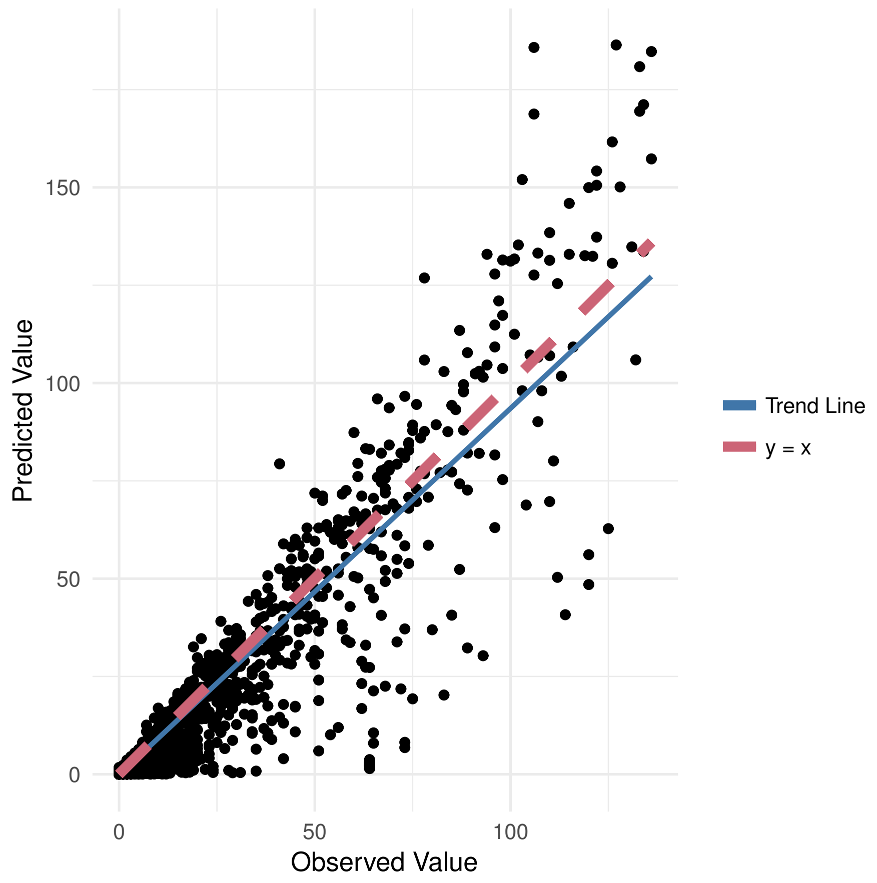


Fig. 3: Predicted vs. observed values.

we see a positive coefficient for the zero component. This is puzzling at first, but may be explained thusly: it is known that contributions are extremely important in order to gain technical trust in OSS [26], supported also by the large coefficient for commits in the zero component (0.453). As the number of likely buggy commits is correlated with the number of overall commits by a developer, this positive coefficient indicates that contributing at all, regardless if one's contribution is buggy, is important in getting the first call mention, and thus the first @-mention.

Productivity In both the zero and count components, we see a positive coefficient for commits, indicating that increased productivity is associated with higher @-mentions. The zero model coefficient is very high. This is in support of productivity being important in receiving the first @-mention.

Responsiveness Interestingly and contrary to our hypothesis, for the count component, we see an insignificant coefficient. Responsiveness is not considered in the zero component as one must be called in order to reply, which means responsiveness is undefined for those with an @-mention count of 0.

Research Answer 1: *We see a positive effect of visibility measured by $\mathcal{OSS}_\rho$, and a negative effect for $\mathcal{ISS}_\kappa$. More likely buggy commits (a measure of negative expertise) is associated with lower @-mentions when one has already been @-mentioned, and higher @-mentions if one has not yet been @-mentioned, possibly explained by the idea that any productivity associates with a first @-mention. We see positive effects for productivity, and no significant effect of responsiveness.*

5.3 Case Study: Attributes of Interest and Model Fit

To further examine **RQ 1** and provide concrete reasoning behind our model's fit, we performed case studies. Specifically, we looked at those with high observed future @mentions but low model predictions, and those who transition from zero to nonzero @-mentions.

5.3.1 Sub-Case Study: High Observed @-Mentions, Low Predicted @-Mentions

For this study, we manually examined those with less than 50 and greater than 15 observed *future* @-mentions, nonzero observed *past* @-mentions, and a predicted @-mention count of less than or equal to 1; *i.e.,* those along the bottom of the x-axis of Fig. 3. In this region, all individuals have never explicitly replied to another developer (*i.e.*, $\mathcal{OSS}_\rho$ and social outdegree are both 0), a low number of commits (1 to 9); as these coefficients are positive in our model, these individuals should be pushed to higher counts. However, all developers in this region also have relatively high $\mathcal{ISS}_\kappa$ (0.1 to 1.0), and have experience in other projects (indicated by a large developer age). As both $\mathcal{ISS}_\kappa$ and developer age have a relatively large negative influence in our model, this explains why our predicted future @-mentions are low from a statistical standpoint.

To dig deeper, we consider the case of a particular developer in this region: developer *arthurevans*, for project *google/WebFundamentals.* In issue #4928 of the project, a discussion about PRPL patterns[14], the poster says: "*I'll defer to the grand master of all things PRPL, @arthurevans for what the final IA for this section might look like*". Although *arthurevans* has low observed activity in the project itself (*e.g.*, low social outdegree and low commit count), this indicates that the poster greatly values *arthurevans*'s input. The story is similar for the others in this region [15]; the issue poster values the opinion of the called-in person, indicating a level of outside-project expertise.

In summary, it appears this region consists of those who are actually expert, but this expertise is not reflected by their in-project contributions. Although

[14] `https://developers.google.com/web/fundamentals/performance/prpl-pattern/`

[15] We could not perform this in-depth study for discussions not in English.

we attempt to capture outside expertise through a developer's overall GitHub age, we were unable to include other metrics of outside expertise (*e.g.*, number of public repositories contributed to) due to high multicollinearity. Orthogonal metrics of outside expertise may exist that can better fit these individuals.

5.3.2 Sub-Case Study: Transitioning From Zero @-Mentions

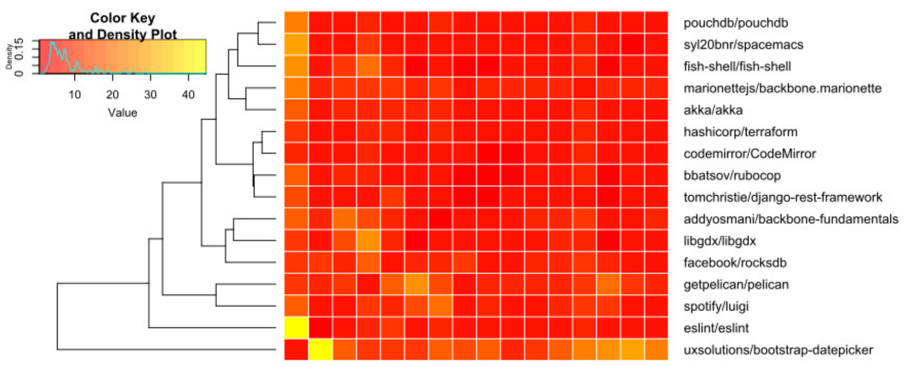
For this study, we took a random sample of 10 individuals (out of 235) who had zero observed @-mentions, but transitioned to nonzero @-mentions in the next 6 months, *i.e.*, our future period. In this region, we observe a combination of factors: project age and newcomers who wish to participate more. Some projects are relatively new or newly popular, which means that although they are rapidly gaining popularity on GitHub, their issue production rate hasn't yet caught up. Though all individuals have contributed to the project, there has not been a chance for @-mentions to be observed; those transitioning from zero @-mentions to nonzero @-mentions would likely have nonzero @-mentions had the observation time split been later in the project.

Perhaps more interesting, we see some new individuals that have recently contributed commits and seem genuinely interested in participating more. For example, in pull request #2587 of the project *prometheus/prometheus*, we see the first call to developer *mattbostock*, causing a transition from zero to nonzero @-mentions. Prior to this, we see that *mattbostock* had been contributing to issue discussions (*e.g.* issues #1983 and #10), bringing up problems and providing potential solutions. Thus, due to signaling interest and participating in discussions (visibility), providing commits (productivity), and having no issues against these commits (expertise), we see them being eventually recognized in an @-mention.

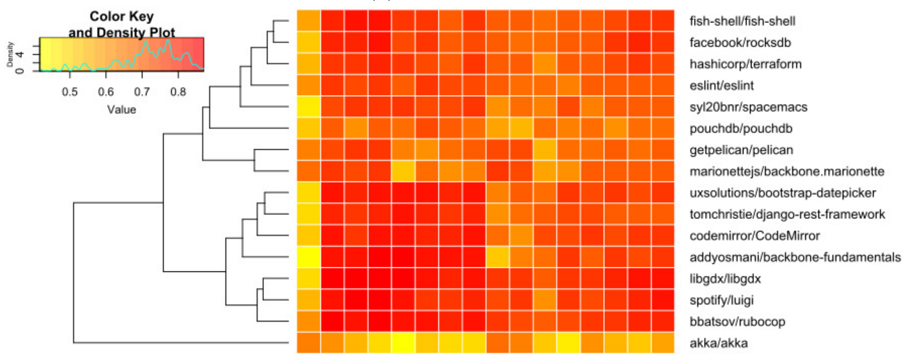
RQ 2: Can models trained entirely on one project be reliably used to predict @-mentions on another project?

To answer this question, we require project-specific models of @-mentions. Due to the sparseness of data, adding a factor to the existing model in Table 2 causes estimation to diverge. Thus, to avoid divergence, we fit simplified models with selected attributes of visibility ($\mathcal{OSS}_\rho$, $\mathcal{ISS}_\kappa$, social outdegree), expertise (likely buggy commits), productivity (commits), responsiveness, and developer's outside project experience (GitHub age). A subset is required due to the smaller number of observations per project; too many variables for too little data can cause issues as, *e.g.*, small multicollinearity can cause big issues for small data. Thus, we select only a few representative variables from each of our groups of interest. For consistency, we explicitly fit separate models for the transition from zero to nonzero (zero component) and for nonzero count (count component), as is done implicitly by the hurdle model.

Fig. 4 contains symmetric heatmaps of predictability for our project-specific models (count and zero, respectively). To measure predictability of the count component, we use the average of *mean absolute error* (MAE) between each pair of models. For projects i and j, with data d_i and d_j, and models y_i and y_j, we compute predicted values $\hat{y}_i = y_i(d_j)$ and $\hat{y}_j = y_j(d_i)$; *i.e.*, we predict



(a) Count model.



(b) Zero model.

Fig. 4: Cross-project predictive power heatmap for each project-specific model, count (a) and zero (b) components.

using one model's fit and the other model's data, thus providing a measure of cross-project model fit. We then compute the average MAE between the two fits *i.e.*, $\frac{\hat{y}_i+\hat{y}_j}{2}$, and plot this value in each heatmap cell. For the zero component, we analogously compute fit by calculating the average *area under the receiver operating characteristic curve* (AUC) between two projects *i.e.*, $\frac{AUC(\hat{y}_i)+AUC(\hat{y}_j)}{2}$. For MAE, a lower value is better; for AUC, a higher value. We then plot a dendrogram, showing clusters of projects based on predictive ability.

For both the count and zero components, we generally see good fit across projects (lower average MAE, higher average AUC), with some outliers. For the count case, we see that *uxsolutions/bootstrap-datepicker* is an anomaly in having poor fit for many projects, being grouped in its own cluster. Otherwise, there are no immediately clear clustering relationships between projects, other than that the mean MAE is generally below 10, as noted in the density plot.

For the zero case, we also see one clear outlier: *akka/akka*. In general, cross-project fits for this project are relatively poor compared to the majority. The reason for this may be due to the difference in importance for our determinants of interest as compared to other projects. Fig. 5 shows our fitted coefficients for each project model. For the zero component, though *akka/akka* does not lie on its own according to hierarchical clustering, we see that its coefficients are very different from other projects, with a negative coefficient for commits and almost zero coefficients for all other variables (except social outdegree). This explains the poor cross-project fit; in this project, a higher number of commits associates with a lower predicted @-mention count, while in the majority of other projects this coefficient is positive (or nearly zero).

In summary, we do see a general trend of good fit for both the count component and, to a lesser extent, the zero component.

Research Answer 2: *The count component of each project-specific model has overall good fit when predicting purely cross-project. A similar trend exists for the zero component, though to a lesser extent on average.*

RQ 3: Is there evidence of project-specific @-mention culture? Or are the determinants of @-mentions a GitHub-wide phenomenon?

Fig. 5 contains heatmaps of coefficients for the count and zero components of our project-specific models. When looking at each column, we see some coefficients that are almost uniformly the same, *e.g.*, responsiveness for both components, commits for the count component, and likely buggy commits for the zero component. However, we do see differences, *e.g.*, $\mathcal{OSS}_\rho$ in both model components is negative for some and positive for other rows.

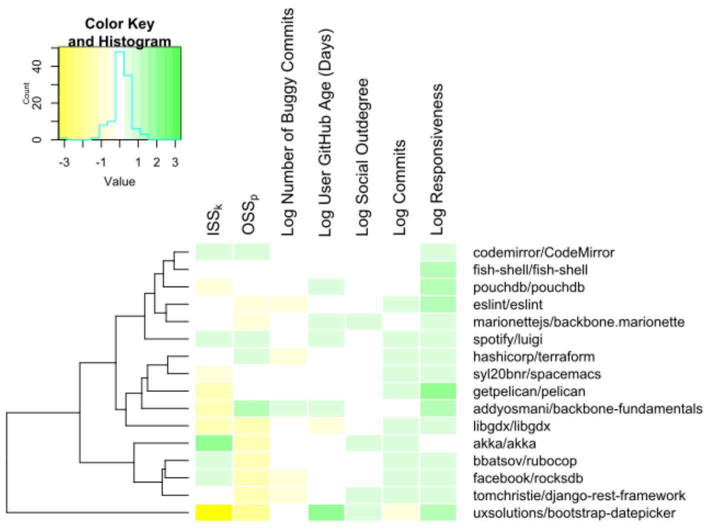
The fact that there are differences per column (*i.e.*, per coefficient) for most coefficients lends credence to the idea that there are project-specific @-mention culture differences on a per attribute basis. However, there are things that don't change across projects, *e.g.*, the importance of commits in gaining more @-mentions. In addition, the generally high cross-project predictive power shown in Fig. 4 suggests that project-specific culture differences may not matter too much. To identify some concrete reasoning behind these particular differences in variable importance, we turn to another case study.

5.4 Case Study: Project-Level Differences

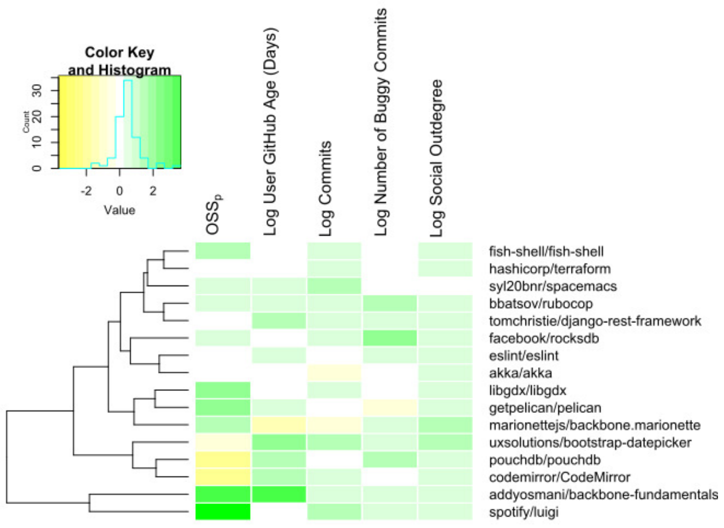
Reflecting on Fig. 5, here we ask: why are some coefficients positive for a number of projects, and negative for others?

As $\mathcal{OSS}_\rho$ seems to exhibit this behavior in both our count and zero models, and significantly so for our global model, we choose it for our study. For the zero model, we see a negative coefficient for projects *uxsolutions/bootstrap-datepicker, pouchdb/pouchdb*, and *codemirror/CodeMirror*; indicating higher specialization in one's replies associates with lower future @-mentions in the projects.

One explanation for this phenomenon could be due to a larger inner circle as compared to other projects; *i.e.*, to gain @-mentions one must become visible to



(a) Count model.



(b) Zero model.

Fig. 5: Heatmap of coefficients for project-specific models, (a) count and (b) zero components.

more people. For both *uxsolutions/bootstrap-datepicker* and *pouchdb/pouchdb*, this seems to be the case. When looking at the distribution of commits across contributors, in both projects the original top committer has largely reduced their commit rate, while in the mean time the second largest committer has picked up the pace. In addition, the distribution of commits seems to be comparatively more uniform across contributors, indicating a larger inner circle.

For *codemirror/CodeMirror*, the distribution of commits is highly concentrated in the top committer; however, when viewing issues, we see that multiple others contribute to review and discussion. This likewise indicates a larger inner circle that one must be visible to. For the count model, the story seems to be the same for projects with a negative coefficient; there is either a more uniform distribution of commits across the top contributors, or a larger number of individuals participating in issue discussions, indicating a larger inner circle.

For projects with positive coefficients, we see a different behavior. In pull requests, it appears the top project members are more open to calling on others to provide input. *E.g.,* for project *spotify/luigi* pull request #2186, a top contributor asks the original poster to run *git blame* on the modified code to see who originally posted it, admitting a lack of expertise about the associated module; we see similar behavior for pull request #2185. For project *addyosmani/backbone-fundamentals* issue #517, we see the project owner calls on another contributor for their input, stating "*[I] would love to suggest your project to devs ...*". Recall that a positive coefficient for $\mathcal{OSS}_\rho$ indicates a specialization in reply behavior, *i.e.*, more focus in one's social behavior. As the top contributors for these projects seem to be the ones calling on others, it appears one may specialize their social behavior towards the top contributors to get noticed; hence, more social specialization may associate with higher future @-mentions.

Research Answer 3: *We see slight indications of project-specific @-mention culture. The high cross-project performance suggests that these differences may not matter much for predictivity.*

6 Practical Implications

Understanding which of the observable attributes of a developer are most correlated with their status can inform developers about how others see them, and help them learn the community values. To that end, our models of future @-mentions based on past observable behavior can be useful for informing developers as to which of their external characteristics matter to others, for them to be called for help. Such an understanding can be a first step to becoming a part of a community, or belonging. In practice, this means examining the coefficients of our models, and noting the attributes (*i.e.*, metrics) having significant positive and significant negative coefficients in the models. Those would be the attributes that the data says may matter most to the community. Since we model two separate phenomena, the count and the zero models in Table 2, there are different attributes that matter to the community for attaining a higher count than for getting the first @-call, as discussed in the Results and Discussion section.

But understanding community values can also lead to wanting to enhance those values, or perhaps modify them. Periodic re-fitting and examination of the models can reveal trajectory changes in the community values, e.g., increased emphasis on participating in discussions, or decreased emphasis on responsiveness over time. If the resulting trajectories are showing a departure from the community goals, then concerted, community-wide efforts, e.g. by establishing guidelines or even rules, may be needed to modify those trajectories and bring them closer to the ideals.

As discussed in Theory and Related Work above (Section 2), the (call) @-mention represents a deeper underlying phenomenon than just a tool for getting attention. It is also a belief in the @-mentionee's abilities to provide valuable input [59], often explicitly stated as being greater than the caller's, to accomplish a particular task at hand. Attaining a social (or technical) status as the developer who "gets the job done" can be desirable, as these individuals may more easily enter the "circle of trust" within a given OSS project [26], potentially reducing the time it takes for their commits to be incorporated into the code base [20,13]. Having higher status also increases one's visibility, and with it the chances of participating (or furthering participation) in popular projects [20]. In addition, social interaction between work group members has been found to be important for task success [53], and the elevation of members to a "role model" status [3,4]. On GitHub, these role models ("rockstars") have been found to be important influencers on the general community, providing examples of coding best practices and facilitating novice developers to learn [20,39]. Our models in Table 2 offer good predictivity, and indicate which attributes of developer activities may be influential in attaining future @-mentions. For example, our model shows that committing more is associated with more future @-mentions, for the first mention and beyond. In addition, having a larger outgoing social presence (social outdegree) is associated with more future @-mentions. Thus, developers who wish to attain the above mentioned status roles can use these model associations as guidelines to finely calibrate their behavior over time to exhibit in the community the characteristics that can result in them gaining higher status.

Our finding that the models can perform cross-project prediction well suggests that, on average, it can be expected that one's activity pattern in a project, if emulated in other projects, can result in the same level of @-mentions for them. This also implies that by simply joining the other project and continuing the same behavior as in the former project one can expect a similar @-mention levels there. However, in each project a developer needs to demonstrate sufficient levels of the predictors mentioned above in order to be called upon, which presumably will take some time.

Some of our results were less obvious than others, *e.g.*, the insignificant effect of responsiveness. This may indicate that it is worth calling on and waiting for the high-status people to get involved, even if they are slow.

From a security perspective, trusting new people with the project's code is associated with more maintenance and supervision, which is certainly a concern. Based on our results in this paper, increased efforts could be useful towards training new people to the specifics of the project's code, *e.g.*, by creating a portal for newcomers [57]. Future work may include building online tools to facilitate newcomer onboarding, *e.g.*, the creation of "tag profiles" which provide suggestions to new users regarding how to increase their @-

mentions, an indicator that they are important to a project's success, thus benefiting the project as a whole; also (and perhaps more controversially) it might be helpful to have tools to measure how often each developer's changes induce future fixes.

7 Threats to Validity

There were challenges involved in all aspects of the work, largely due to the loaded reasoning behind @-mentions. Being @-mentioned is not just a result of technical prowess; @-mentioning is also a social phenomenon. Many potential issues were anticipated and carefully addressed. Once we settled on the idea of using call @-mentions, we were able to connect our outcome with background theory on the multidimensionality of @-mentions. To define @-mentions precisely, we necessarily had to narrow our definition specifically to call mentions in issue discussions.

We acknowledge that only considering individuals with non-zero commits is a threat. However, the density of @-mentions for those with zero commits is highly concentrated at zero (median 0 @-mentions for those with zero commits, mean 0.26). In addition, this helps alleviate the threat of bots as bots often must be @-mentioned to be activated; however, most bots do not show up in the "authors" field for a given commit (they may show up in the "committer" field, but we do not use this field in our work). In addition, as we have data from many projects, unless bots are the vast majority of our data points, they should not have a palpable effect on our model fit. We also manually inspected commits to try and filter out bots. However, we cannot be certain that all bots are removed from our data; thus, we acknowledge bots as a threat.

Regarding our identification of "reply" and "call" @-mentions, it is possible that we misidentify a "reply" as a "call" if an individual uses multiple GitHub accounts to post within a project. *E.g.*, if someone first posts with username A, then later posts as username B, we would identify B as a "call" rather than a "reply". Due to the confusion that would likely ensue if this was common, we do not see this as a major threat.

We acknowledge that our identification of likely buggy commits may have issues. The method we use is supported by multiple prior works [55, 37], and seen as a "standard method". However, this does not have the robustness of a method that, *e.g.*, uses an explicit bug tracker to identify buggy commits, and has issues as described by prior work [19, 52]. Due to the structure of GitHub, there is no guaranteed method to identify buggy commits that will work across all projects. Thus, we acknowledge this as a threat.

We note that our operationalization of productivity through commits has its issues. This includes the issue of varying commit styles amongst developers and between projects; some projects may want commits to be as small as possible, while others may not care for or enforce such a rule. We also tested for the inclusion of $\mathcal{MAF}$ and $\mathcal{DAF}$ in their original forms from prior work, which can be seen as productivity measures, as well as raw lines of code.

However, these measures were highly correlated with commits. Thus, we chose to represent productivity with a more easily understood measure - commits - for the sake of model parsimony [60].

Though we attempt to model developer expertise through their GitHub age (in days), we acknowledge a threat with this operationalization. Though age is often an important indicator of expertise, expertise can be measured outside of age, *e.g.*, a contributor may have a specific skill set that is sought after in a project, and thus may be called upon regardless of their age. We do have a semi-overlapping measurement of code-based skill through commits, though this is not guaranteed to be adequate in controlling for this phenomenon. And, as noted above, $\mathcal{MAF}$ and $\mathcal{DAF}$ - which can be interpreted as more complex measures of expertise - were highly correlated with commits. It may be possible to measure skill-specific expertise by, *e.g.*, looking at a developer's commit history and seeing the subject domain of their contributions prior to being called. However, this measurement would be difficult to obtain as it would require a labeling of subject domains, which GitHub does not reliably provide. Thus, though we attempt to control for developer expertise, we note this as a threat to validity.

We also acknowledge that coefficient estimates are somewhat small, and we do not report effect sizes (beyond interpreting coefficient estimates as "effect sizes" themselves). This is because standard effect size calculations, *e.g.,* Cohen's d [17], are not well-behaved for non-Gaussian distributions - as is the case in our models.

Our case studies would benefit from larger amount of data. The case study sizes were due to the regions of interest; our regions were small, and thus our case studies were relatively small.

Our work is supported by prior qualitative research into @-mention usage. Still, we acknowledge that our study would likely benefit from further qualitative studies, *e.g.*, a survey of developers on their use of the @-mention.

8 Conclusion

We performed a quantitative study of @-mentions in GitHub, as captured in calls to people in discussions. We supplemented those with case studies on samples of discussions, to help triangulate our findings. Our models have good fits to the data, suggesting that our formulation of @-mentions is explained well by the data.

The idea that projects in an ecosystem have similar models of what it means to be worthy of an @-mention is appealing. We find that the good cross-project predictive power cannot be simply distilled down to productivity in our models, thus adding evidence toward the multidimensional nature of @-mentions. It is also very reasonable that there would be cliques of projects in which the sense of who to @-mention is even more uniform than across the whole ecosystem, and our findings underscore that. Obvious open questions here are: how do notions of @-mentions get in sync? And, to borrow from ecology, does the robustness of the @-mention models across GitHub convey any fitness benefit in the ecosystem? We can see a plausible mechanism that would offer an answer to the first: projects share people and people cross-pollinate the @-mentioning behavior across projects in which they participate. We leave the validation of this, and other models, to future work. The @-mention model robustness, likewise, implies some preference for success, be it by design or an emerging one, across the ecosystem. This can be a function of people's mobility in the ecosystem and their preference for and vigilance to participate in popular projects; we leave the answers for future work.

References

1. Ackerman, A.F., Fowler, P.J., Ebenau, R.G.: Software inspections and the industrial production of software. In: Proc. of a symposium on Software validation: inspection-testing-verification-alternatives, pp. 13–40. Elsevier North-Holland, Inc. (1984)
2. Allison, P.: When can you safely ignore multicollinearity? `https://statisticalhorizons.com/multicollinearity` (2012)
3. Bandura, A.: Aggression: A social learning analysis. Prentice-Hall (1973)
4. Bandura, A., Walters, R.H.: Social learning theory (1977)
5. Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the royal statistical society. Series B (Methodological) pp. 289–300 (1995)
6. Bird, C., Gourley, A., Devanbu, P., Swaminathan, A., Hsu, G.: Open borders? immigration in open source projects. In: The Fourth International Workshop on Mining Software Repositories (2007)
7. Bird, C., Rigby, P.C., Barr, E.T., Hamilton, D.J., German, D.M., Devanbu, P.: The promises and perils of mining git. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on, pp. 1–10. IEEE (2009)
8. Blüthgen, N., Menzel, F., Blüthgen, N.: Measuring specialization in species interaction networks. BMC ecology **6**(1), 9 (2006)
9. Brenkert, G.G.: Trust, business and business ethics: an introduction. Business Ethics Quarterly **8**(2), 195–203 (1998)
10. Brockner, J.: Understanding the interaction between procedural and distributive justice: The role of trust. (1996)
11. Burke, M., Marlow, C., Lento, T.: Feed me: motivating newcomer contribution in social network sites. In: Proceedings of the SIGCHI conference on human factors in computing systems, pp. 945–954. ACM (2009)
12. Burke, M., Marlow, C., Lento, T.: Social network activity and social well-being. In: Proceedings of the SIGCHI conference on human factors in computing systems, pp. 1909–1912. ACM (2010)
13. Calefato, F., Lanubile, F., Novielli, N.: A preliminary analysis on the effects of propensity to trust in distributed software development. In: Global Software Engineering (ICGSE), 2017 IEEE 12th International Conference on, pp. 56–60. IEEE (2017)
14. Cameron, A.C., Trivedi, P.K.: Regression analysis of count data, vol. 53. Cambridge university press (2013)
15. Casalnuovo, C., Vasilescu, B., Devanbu, P., Filkov, V.: Developer onboarding in github: the role of prior social links and language experience. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 817–828. ACM (2015)
16. Chow, S.C., Shao, J., Wang, H., Lokhnygina, Y.: Sample size calculations in clinical research. Chapman and Hall/CRC (2017)
17. Cohen, J.: Statistical power analysis for the behavioural sciences (1988)
18. Cohen, J., Cohen, P., West, S.G., Aiken, L.S.: Applied multiple regression/correlation analysis for the behavioral sciences. Routledge (2013)

19. da Costa, D.A., McIntosh, S., Shang, W., Kulesza, U., Coelho, R., Hassan, A.E.: A framework for evaluating the results of the szz approach for identifying bug-introducing changes. IEEE Transactions on Software Engineering **43**(7), 641–657 (2017)
20. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in github: transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pp. 1277–1286. ACM (2012)
21. Dourish, P., Chalmers, M.: Running out of space: Models of information navigation. In: Short paper presented at HCI, vol. 94, pp. 23–26 (1994)
22. Ducheneaut, N.: Socialization in an open source software community: A socio-technical analysis. Computer Supported Cooperative Work (CSCW) **14**(4), 323–368 (2005)
23. Faraway, J.J.: Linear models with R. CRC press (2014)
24. Gallivan, M.J.: Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. Information Systems Journal **11**(4), 277–304 (2001)
25. Gharehyazie, M., Posnett, D., Filkov, V.: Social activities rival patch submission for prediction of developer initiation in oss projects. In: Software Maintenance (ICSM), 2013 29th IEEE International Conference on, pp. 340–349. IEEE (2013)
26. Gharehyazie, M., Posnett, D., Vasilescu, B., Filkov, V.: Developer initiation and social interactions in oss: A case study of the apache software foundation. Empirical Software Engineering **20**(5), 1318–1353 (2015)
27. Good, I.J.: The population frequencies of species and the estimation of population parameters. Biometrika **40**(3-4), 237–264 (1953)
28. Handy, C.: Trust and the virtual organization. Harvard business review **73**(3), 40–51 (1995)
29. Hossain, L., Zhu, D.: Social networks and coordination performance of distributed software development teams. The Journal of High Technology Management Research **20**(1), 52–61 (2009)
30. Husted, B.W.: The ethical limits of trust in business relations. Business Ethics Quarterly **8**(2), 233–248 (1998)
31. Ibrahim, W.M., Bettenburg, N., Shihab, E., Adams, B., Hassan, A.E.: Should i contribute to this discussion? In: Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, pp. 181–190. IEEE (2010)
32. Inglehan, R.: Trust, well-being and democracy. Democracy and trust p. 88 (1999)
33. Jarvenpaa, S.L., Knoll, K., Leidner, D.E.: Is anybody out there? antecedents of trust in global virtual teams. Journal of management information systems **14**(4), 29–64 (1998)
34. Jones, T.M., Bowie, N.E.: Moral hazards on the road to the virtual corporation. Business Ethics Quarterly **8**(2), 273–292 (1998)
35. Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., German, D.M.: Open source-style collaborative development practices in commercial projects using github. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, pp. 574–585. IEEE Press (2015)
36. Kavaler, D., Sirovica, S., Hellendoorn, V., Aranovich, R., Filkov, V.: Perceived language complexity in github issue discussions and their effect on issue resolution. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 72–83. IEEE Press (2017)
37. Kim, S., Zimmermann, T., Pan, K., James Jr, E., et al.: Automatic identification of bug-introducing changes. In: Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on, pp. 81–90. IEEE (2006)
38. Kramer, R.M., Tyler, T.R.: Trust in organizations: Frontiers of theory and research. Sage (1996)
39. Lee, M.J., Ferwerda, B., Choi, J., Hahn, J., Moon, J.Y., Kim, J.: Github developers use rockstars to overcome overflow of news. In: CHI'13 Extended Abstracts on Human Factors in Computing Systems, pp. 133–138. ACM (2013)
40. Matter, D., Kuhn, A., Nierstrasz, O.: Assigning bug reports using a vocabulary-based expertise model of developers. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on, pp. 131–140. IEEE (2009)
41. McDonald, N., Goggins, S.: Performance and participation in open source software on github. In: CHI'13 Extended Abstracts on Human Factors in Computing Systems, pp. 139–144. ACM (2013)

42. McKnight, D.H., Choudhury, V., Kacmar, C.: Developing and validating trust measures for e-commerce: An integrative typology. Information systems research **13**(3), 334–359 (2002)
43. Mockus, A., Herbsleb, J.D.: Expertise browser: a quantitative approach to identifying expertise. In: Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on, pp. 503–512. IEEE (2002)
44. Murphy, G., Cubranic, D.: Automatic bug triage using text categorization. In: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. Citeseer (2004)
45. Newton, K.: Trust, social capital, civil society, and democracy. International Political Science Review **22**(2), 201–214 (2001)
46. Oeldorf-Hirsch, A., Sundar, S.S.: Posting, commenting, and tagging: Effects of sharing news stories on facebook. Computers in Human Behavior **44**, 240–249 (2015)
47. O'Leary, M., Orlikowski, W., Yates, J.: Distributed work over the centuries: Trust and control in the hudson's bay company, 1670-1826. Distributed work pp. 27–54 (2002)
48. Posnett, D., D'Souza, R., Devanbu, P., Filkov, V.: Dual ecological measures of focus in software development. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 452–461. IEEE Press (2013)
49. Qiu, L., Lin, H., Leung, A.K.y.: Cultural differences and switching of in-group sharing behavior between an american (facebook) and a chinese (renren) social networking site. Journal of Cross-Cultural Psychology **44**(1), 106–121 (2013)
50. Robert, L.P., Denis, A.R., Hung, Y.T.C.: Individual swift trust and knowledge-based trust in face-to-face and virtual team members. Journal of Management Information Systems **26**(2), 241–279 (2009)
51. Rodrıguez, G.: Models for count data with overdispersion (2013)
52. Rodríguez-Pérez, G., Zaidman, A., Serebrenik, A., Robles, G., González-Barahona, J.M.: What if a bug has a different origin? making sense of bugs without an explicit bug introducing change. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, p. 52. ACM (2018)
53. Saavedra, R., Earley, P.C., Van Dyne, L.: Complex interdependence in task-performing groups. Journal of applied psychology **78**(1), 61 (1993)
54. Sato, Y., Arita, S.: Impact of globalization on social mobility in japan and korea: Focusing on middle classes in fluid societies. International Journal of Japanese Sociology **13**(1), 36–52 (2004)
55. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: ACM sigsoft software engineering notes, vol. 30, pp. 1–5. ACM (2005)
56. Steinmacher, I., Conte, T., Gerosa, M.A., Redmiles, D.: Social barriers faced by newcomers placing their first contribution in open source software projects. In: Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing, pp. 1379–1392. ACM (2015)
57. Steinmacher, I., Conte, T.U., Treude, C., Gerosa, M.A.: Overcoming open source project entry barriers with a portal for newcomers. In: International Conference on Software Engineering (2016)
58. Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., Taylor, P., Martin, R., Van Ess-Dykema, C., Meteer, M.: Dialogue act modeling for automatic tagging and recognition of conversational speech. Computational linguistics **26**(3), 339–373 (2000)
59. Tsay, J., Dabbish, L., Herbsleb, J.: Let's talk about it: evaluating contributions through discussion in github. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering, pp. 144–154. ACM (2014)
60. Vandekerckhove, J., Matzke, D., Wagenmakers, E.J.: Model comparison and the principle. In: The Oxford handbook of computational and mathematical psychology, vol. 300. Oxford Library of Psychology (2015)
61. Vuong, Q.H.: Likelihood ratio tests for model selection and non-nested hypotheses. Econometrica: Journal of the Econometric Society pp. 307–333 (1989)
62. Yu, Y., Wang, H., Yin, G., Wang, T.: Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? Information and Software Technology **74**, 204–218 (2016)

63. Yu, Y., Yin, G., Wang, H., Wang, T.: Exploring the patterns of social behavior in github. In: Proceedings of the 1st international workshop on crowd-based software development methods and technologies, pp. 31–36. ACM (2014)
64. Zhang, Y., Wang, H., Yin, G., Wang, T., Yu, Y.: Exploring the use of@-mention to assist software development in github. In: Proceedings of the 7th Asia-Pacific Symposium on Internetware, pp. 83–92. ACM (2015)
65. Zhang, Y., Wang, H., Yin, G., Wang, T., Yu, Y.: Social media in github: the role of @-mention in assisting software development. Science China Information Sciences **60**(3), 032102 (2017)
66. Zucker, L.G.: Production of trust: Institutional sources of economic structure, 1840–1920. Research in organizational behavior (1986)