Elements of Socio-Technical Trust in GitHub

The Effects of Visibility, Expertise, Productivity, and Responsiveness

Anonymous Author(s)

ABSTRACT

Open Source Software (OSS) supports dynamic teams across a wide variety of social and technical backgrounds. OSS project success relies on crowd contributions; though a small number of developers are primary contributors, for tasks such as help with issue identification and documentation, and bug fixing, minority contributors are also called on. It is, then, important to know who can help and who can be trusted with important task-related duties, and why.

In this paper, we argue that @-mentions in GitHub issues and pull request discussions can be appropriately used as signals of trust. We built overall and project-specific predictive future trust models of @-mentions, in order to capture the determinants of trust in each of two hundred 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 trust, while responsiveness is not, when controlling for confounds. Also, we find that even though project-specific differences exist in the trust models, the overall model can be used for cross-project prediction, indicating its GitHub-wide viability and utility.

ACM Reference Format:
Anonymous Author(s). 2017. Elements of Socio-Technical Trust in GitHub. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 12 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Open-source software projects are examples of socio-technical systems [23, 46] whose operation is influenced by both team social structure, and the technical design of the software [4, 49]. The links in the social network arise from ongoing interactions among developers, including communication, collaborations, etc, These networks are *task-oriented*: links emerge, endure or vanish according to tasks undertaken by developers, and their perceived needs for interactions with other developers they consider relevant to those tasks. The importance of these social links in the context of open-source and other software projects has been well-acknowledged and widely studied, since the early days of open-source projects [2, 5, 15, 24, 45]. Social links have been considered both as markers of social status [45] and as indicators of task success & productivity [2]. A developer linking to another generally indicates trust in that person's abilities or knowledge. E.g., a developer who had written most of the DB portion of an API is probably the right person to trust with questions about that API functionality, and call on as need arises. Our goal in this work is study the factors that are associated with the emergence of growth of this kind of trust.

In modern, social-coding [16] projects based on sites like GitHub and Bitbucket, that favour the pull-request model, the emergence and growth of trust can be explicitly observed in task-oriented technical discussions. On GitHub, the @-mention in issue discussions 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, one can consider the network of @-mentions, or calls, as a sort of a directed social network, with a task-oriented purpose. These mentions are heavily used; in our data, 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; excluding those who never respond[1], developers respond 42.94% of the time. @-mention popularity reflects the central role they play in task-oriented social interactions. If a person is referenced many times across discussions, it is likely they are perceived by many as helpful in many different tasks. Such trusted developers can have outsized roles and responsibilities in the project network. Most of such reputations are no doubt deserved, but some may not be. Naturally, when things go wrong, existing trust can be reassessed, but it may be too late. Hence the need to understand the elements, or features, of technical trust. When is trust in a developer deserved? Who do we trust to answer our question? Why do we trust someone?

Trust is a multidimensional phenomenon [18, 35], with a long-recognized social component and well understood benefits to social and economical well-being [27, 36], in both physical and virtual teams [28]. While individuals do have a personal notion of when to trust someone, in social setting those notions inherit from the communal sense of trust [27, 28, 36]. In socio-technical groups like software projects, contributors must be trusted as technically competent, and also as useful to the project. Previous work has explored the factors associated with the gaining of contributor status in open-source projects which operate *with a centralized repository*. In such settings, the ability to write to the central repository must be controlled carefully, and thus gaining contributor status is a key indicator of trust worth careful study. Considerable prior work has done so [3, 13, 17, 19, 43].

In pull-request oriented models, with *decentralized* repositories, code contributions can be made to forks, without restraint. Anyone can create forks, make changes, and submit "pull-requests" for these contributions to be incorporated. Here, social processes such as code-review take a central role in deciding the fate of code

[1]Those who never respond are often, *e.g.*, developers of an upstream library who are not active in the downstream project.

contributions. Opinions from trusted people during the relevant discussions would be in great demand, and thus, the social demands 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 of technical trust, i.e., technical trust may be a global, or cross-project phenomenon.

The goal of this paper is to understand both the elements of trust and the extent to which the elements of trust are in common to projects across GitHub. A key insight enabling this study is our use of @-mentions in discussions as a proxy for trust in GitHub projects. Starting from data from 200 GitHub projects on @-mentions and comprehensive developer and project metrics, we sought to quantitatively model future trust predictively, from past observations of developers' visibility, expertise, productivity, and responsiveness in the projects. From our models, together with case studies aimed and triangulating the model results, we found the following:

- We review existing trust theory in sociology, psychology, and management, and its relationship to GitHub. We find that we can mine a reliable trust signal from GitHub as per existing theoretical definitions, with slight modifications to accommodate the specifics of GitHub.
- We see a net positive effect of visibility on trust. We see that more buggy commits (*i.e.*, negative expertise) leads to lower trust when one is already trusted, and higher trust if one is not already trusted; perhaps explained by the idea that any productivity, even buggy, leads to an initial trust extension. We see positive effects for productivity, and no significant effect of responsiveness.
- We see that purely cross-project model fits are generally good. For gaining trust beyond the first extension, the fit is comparatively better on average than models for gaining a first trust extension.
- We see indications of project-specific trust culture, however, generally high cross-project performance suggests these differences may not matter much, especially in prediction.

In the following, we present the theoretical underpinnings and our research questions in Section 2, and the data and methods in Section 3. Section 4 presents the results and discussion, including case studies, and Section 5 the threats to validity and conclusion.

2 THEORY AND RELATED WORK

To understand the notion and elements of trust in OSS projects, we build a theory that draws from existing OSS literature, as well as sociological theory on tagging and mentioning behavior. First, we introduce our definition of trust as based on work done in the fields of sociology, psychology, and management. We then introduce our trust signal on GitHub, supported by prior work. We then discuss the importance of social exchange and interaction on OSS project success.

Trust in Context Oft-mentioned and widely discussed, the meaning and role of trust has been examined across many disciplines, including sociology, psychology, and philosophy. Each discipline views trust in a different light; if one asks a room of people what trust means, one will likely get many different, but overlapping, answers - which makes studying trust challenging. Still, as we study the concept of trust and its affecters in GitHub, we must define trust specifically.

Summaries of existing trust literature across varied fields propose different taxonomies of trust [9, 33, 54]. One prevailing shared notion is that the truster must place themselves in "a position of vulnerability to or risk from the trustee"; *i.e.*, the truster knowingly makes themselves vulnerable to potentially malicious behavior by the trustee, regardless of the truster's ability to monitor the trustee's behavior [8, 26]. It is generally agreed that this condition is the key component to any definition of trust. Gallivan provides a succinct set of definitions for trust types as provided by prior work on organizational trust, reiterated here [18]:

- *Knowledge-based trust*: trust based upon a prior history of transactions between two parties.
- *Characteristic-based trust*: trust that is assumed, based on certain attributes of the other party.
- *Institutional-based trust* - a trusting environment, as ensured by guarantor agencies.
- *Justice-based trust*: related to the concept of procedural justice (*i.e.*, ensuring fair procedures).
- *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 in GitHub. Jones and Bowie state that "the efficiency of [virtual teams] depends on features - speed and flexibility - that require high levels of mutual trust and cooperation" [30]. O'Leary *et al.* state, in reference to distributed work: "... trust is described as critical ... because it is impossible to monitor and control geographically distributed employees" [38]. Both authors cite Handy, who was among the first to examine trust in virtual organizations, stating: "... the managerial dilemma comes down to the question, How do you manage people whom you do not see? The simple answer is, by trusting them ..." [22].

Though swift trust may initially appear most applicable by the concise descriptions above, much of the founding work was done in the 1990s, prior to the proliferation of socio-technical systems such as GitHub. As such, a sweeping categorization of GitHub as having a swift trust system is likely incomplete. There are aspects of multiple trust frameworks, as described above, that may more fully apply to GitHub. Knowledge-based trust likely plays a part; extended contact in long-lasting projects happens often. Characteristic-based trust is also likely; developer task characteristics can be easily seen on GitHub, *e.g.*, number of repositories contributed to, *etc.* Both institutional- and justice-based trust are not applicable here. Thus, we believe a blended definition of trust is more suitable for GitHub.

More recently, Robert *et al.* redefine swift trust for modern systems as "a category-matching process based on team member characteristics" [41], and a combination of classical swift trust and knowledge-based trust, seemingly with parts of characteristic-based trust. It is by this definition that we mostly abide; we believe that trust in GitHub is generated by a belief in the trustee's *readily observable characteristics*, *e.g.*, commit behavior, code quality, project expertise, *etc.*, and that such belief serves to assuage the feeling of risk that the truster incurs. As GitHub is a task-oriented, socio-technical system, we further define trust in GitHub as a belief in the trustee's characteristics in relation to the task at hand; a belief that trustee will be useful in addressing the task. Therefore:

> In this work, we define trust as *an extension* from trustee to truster, generated by the truster's knowledge of the trustee's observable characteristics, due to the truster's belief in the trustee's ability to address a given task.

@: A Signal of Trust in GitHub GitHub projects have issue trackers with a rich feature set, including ticket labeling, milestone tracking, and code tagging. For each project on Github, 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 will receive a notification that they are being referenced in a discussion. This aspect is crucial to our notion of trust. 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 will be useful in addressing the task at hand. To validate the use of @-mentions (specifically *call* @-mentions) as a measure of trust, as defined above, we look to prior literature on the reasons behind use of the @-mention.

Tsay *et al.* performed interviews with several developers of popular projects on GitHub, specifically related to the discussion and evaluation of contributions [47]. 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 [31]. 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 in order to draw other members' attention to a particular problem. In our manual observation of issues for the case studies described in Section 4, we found that @-mentions, specifically those to @-mentionees who are not currently participating in the given discussion, seem to be universally used as a call-for-attention, for a variety of reasons. In many cases, when the @-mentioner is a core project member, the @-mentionee is called to provide input due to a stated lack of knowledge by the @-mentioner for the related code segment at hand, or to draw the attention of other core members for input. Thus, through the qualitative work of others and our own manual case studies, it seems the @-mention signal is consistently extended as trust.

Although GitHub is wildly popular, there is a relative few studies specifically regarding the @-mention and its relationship to project success. Zhang *et al.* performed an initial look into @-mentions and project performance on GitHub, and found that more difficult-to-solve issues (*e.g.*, longer length of discussion) have more @-mentions [53]. In addition, Yu *et al.* found that having @-mentions in a discussion decreases the time to resolve an issue [50]. However, neither Zhang *et al.* nor Yu *et al.* separated @-mentions into the types we do here: *call* and *reply* @-mentions, and our study is thus orthogonal to theirs.[3]

The Importance of 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 across a wide variety of fields in identifying reasons behind social tagging and mentioning behavior, including in GitHub [51].

In the field of psychology, Qiu *et al.* studied the cultural differences in sharing behavior between Facebook and its Chinese counterpart, Renren [40]. They found that the sense of community invoked by participants varies drastically; Facebook users are more closely knit into subcommunities, while Renren users exhibit a larger communal sense of belonging. Also in the field of psychology, Oeldorf *et al.* found that tagging others (analogous to @-mentioning on GitHub) leads to a sense of community, where the motivation to tag lies mostly in seeking others' opinions rather than sharing one's own [37]. Burke *et al.* found that social tagging is a way to increase one's bonding social capital and lower loneliness [11]. These sociological and psychological based findings are of importance to GitHub as they elucidate the importance of social interaction and sense of community in general social interactions, which are known to be important to OSS success [19, 20]. In addition, Burke *et al.* found that those who receive feedback on their Facebook posts and have a wide audience share more [10]. 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.

Of specific importance to GitHub, 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, downloads, *etc* [34]. 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 [25], 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 contrast our work with that of Jeong *et al.* , who created bug tossing graphs in order to improve bug triage [29]. There, the work was focused on determining the best target to "toss" a bug for resolution, based on developer productivity attributes. Here,

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

[3]Described further in Section 3, 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 (81.16%); however, there is a relatively low correlation between call @-mentions and discussion length (28.29%). As our primary focus is on call @-mentions, the correlation between reply @-mentions and discussion length is not a threat for our hypotheses.

we are interested in describing trust - in how social coordinations originate, rather than precise issue resolution.

2.1 Research Questions

We are interested in identifying and examining the effects of observable developer characteristics on our measure of trust: @-mentions which call a developer to a task-oriented discussion. GitHub is a rich resource laden with minable developer characteristics. Specifically, we wish to study the effects of measures known to be important for OSS, in addition to social measures. We describe *visibility* as a social measure, measuring the ability of others to know about the trustee's existence; if one wishes to extend trust, they must have knowledge of the network in order to know who they are capable of reaching. We define expertise by task-related measures, *e.g.*, number of buggy commits, which may theoretically affect the belief a truster has in a potential trustee's ability to address a given task. Productivity is defined by number of commits; those who commit more are likely to 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 trustee is called to lend their talent, it is not farfetched to believe that those who respond to the call more are more likely to be trusted further in the future.

We explicitly model *future* trust, *i.e.*, trust as measured 6 months beyond the "observation period", described further in Section 3. 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 or predict future trust as a function 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 trust on another project?

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

RQ 3: Is there evidence of project-specific trust culture? Or are the affecters of trust a GitHub-wide phenomenon?

3 DATA AND METHODOLOGY

All data was collected by querying GitHub's public API using the Python package PyGithub[4], with the exception of bug data, which was gathered by cloning individual repositories (described below).

Filtering and Cleaning Our data set started as a sample of 200 projects from the top 900 most starred and followed projects. The number of stars and followers are proxies for project popularity, and can identify projects likely to contain enough issues and commits to model trust. As some measures are expensive to calculate, and we wanted a mixture of high-popularity and medium-popularity projects, we decided to start with a 200 project sample to avoid skew towards the upper or lower ends of popularity within the 900.

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 the entirety of the 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 zeroes in our covariates, as many developers participate in issue discussions but never contribute. In addition, this was done in order to focus on those who may become trusted in the future; without any commits, we believe it would be difficult to become trusted as per our definition in Section 2.

As we wish to explicitly model future trust, 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 (trust) 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 then filtered out each individual who had a project participation period of less than 3 months in order to have confidence that their participation has had a chance to stabilize. Thus, we explicitly model future trust levels, as our response period is completely disjoint from our observation period.

In total, this yielded a final pool of 154 unique projects comprised of 17, 171 project-developer pairs to test our hypotheses.

Issues and Trust 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; Figure 1 depicts this

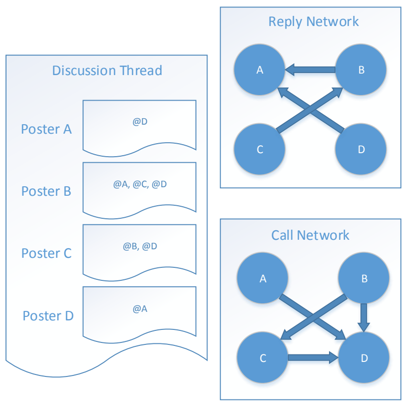

Figure 1: The network creation process. Shown is a discussion thread and the resulting reply and call networks. Note this can be a multigraph (not shown).

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

[5]Specifically, some responses from GitHub's API returned null data that was not handled properly within PyGithub.

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

process. Similar to Zhang *et al.* [52], *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. In addition, 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 a representation of trust as described in Section 2; u calls upon v as u trusts v's input for the discussion at hand.

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. 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 to extend a social (or trust) tie, 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.*. 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 [21], and can be derived from Kullback-Leibler divergence. For discrete probability distributions P and Q, Kullback-Leibler divergence (KL) is defined as:

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

Bluthgen *et al.* define a species diversity measure, δ[8], using D_{KL} [7]. 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) [39]. 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" ($\mathcal{MAF}_j$)[9].

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 the focus developer u gives 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 *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 *outward social specialization* and *inward social specialization* 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.

Attributing Bugs To gather bug data, we implemented the standard SZZ algorithm as defined in [42] and expanded in [32], with a few changes to accommodate the nuances of GitHub. As mentioned above, GitHub has a built-in issue tracking system. In addition, GitHub allows developers to easily close open issues by using a set of keywords[10] in either the body of their pull request or commit message. For example, if a developer creates a fix which addresses issue #123, they can submit a pull request containing the phrase "closes #123"; when the corresponding fixing patch is merged into the repository, issue #123 is closed automatically. Thus, to identify bug-fixing commits, we search for associated issue-closing keywords in all pull requests and commits. We then "git blame" the corresponding fixing lines to identify the last commit(s) that changed the fixing lines, *i.e*, the buggy lines. We assume that the latest change to the fixing lines were those that induced the issue.

We note that an issue is a moderately loose 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 [6].

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

Visibility: We define visibility as the ability for general developers to identify a person's existence; if developer A is not aware of the existence of developer B, there is no way that A would reasonably be able to trust B. Here, we use our social specialization

[7]Though we do use outdegree in our model as well.
[8]This measure is originally called d by Bluthgen *et al.*, but we will use δ here to reserve d to represent developers.
[9]We do not use $\mathcal{MAF}$ directly in our work, but use an analogous form for our social networks.
[10]https://help.github.com/articles/closing-issues-using-keywords/

measures $\mathcal{OSS}_\rho$, $\mathcal{OSS}_\kappa$, and $\mathcal{ISS}_\kappa$, along with total social out-degree (total number of @-mentions for a developer in a given project) as measures of visibility. We believe these measures to be reasonable as they readily identify one's existence within the social network of a given project.

Expertise: We define expertise as a developer's general aptitude to complete project tasks in a manner that is expected by the group. To represent this, we use number of buggy 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 bugs can indicate a lack of aptitude for programming according to the project's goals [11]. It has been shown that a higher $\mathcal{DAF}$ (*i.e.*, higher module specialization) is associated with fewer bugs in a developer's code [39]. Thus, $\mathcal{DAF}$ can represent developer's expertise in particular code modules. The top committer or project owner factor indicates a certain level of prestige and expertise; one would expect that 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 high collinearity with number of bug commits.

Productivity: We measure productivity as the raw commit (authoring) count. There are multiple methods to measure productivity, but most have been shown (of those we computed, *e.g.*, lines of code added or deleted) to be highly correlated with commit count, especially in models where confounds are recognized. We choose commit count as it is the simplest [12].

Responsiveness: 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 a level of care and respect for the caller, 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.

Extra-Project Controls: As stated, our interest is to identify *readily observable* attributes of trust (*e.g.*, within-project social activity and commit activity), and functions thereof, in order to describe and predict developer trust on GitHub. 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 trust level 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 trust 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 the developer, as this is also readibly observable; however, this was highly correlated with age, and was thus dropped from the model.

[11] Note that we use issues as a proxy for bugs, and thus a higher value of this variable does not necessarily mean a lack of aptitude, but still represents a departure from expected coding behavior and thus, expertise.

[12] *I.e.*, Occam's razor.

Modeling Future Trust To seek answers for our research 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 of interest (*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 which model a count response. In our work, we are interested in trust as measured by number of incoming @-mention calls per person - a count. In addition, as our data contain many zeroes, we need a method that can accommodate; the methods listed above all have moderate to severe problems with modeling zeroes. *Zero inflated negative binomial regression* and *hurdle regression* are two methods specifically designed to address this challenge by explicitly modeling the existence of excess zeroes [12]. 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 [48].

We employ *log* transformations to stabilize coefficient estimation and improve model fit, when appropriate [14]. We remove non-control variables that introduce *multicollinearity* measured by *variance inflation factor* > 4 (VIF) [13], as multicollinearity reduces inferential ability; this is below the generally recommended maximum of 5 to 10 [14]. 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 [1]. We model on the person-project level, *i.e.*, each observation is a person within a project.

As noted in Section 2.1, we explicitly model future trust; 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.

4 RESULTS AND DISCUSSION

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

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

RQ 1: Can we describe or predict future trust as a function of developer visibility, expertise, productivity, and responsiveness?

Table 1 shows our model of future trust, with affecters of interest grouped and separated from one another. Figure 3 depcits predicted and observed values along with a $y = x$ and trend line. The mean

[13] *E.g.*, we do not use $\mathcal{ISS}_\rho$ due to high VIF.

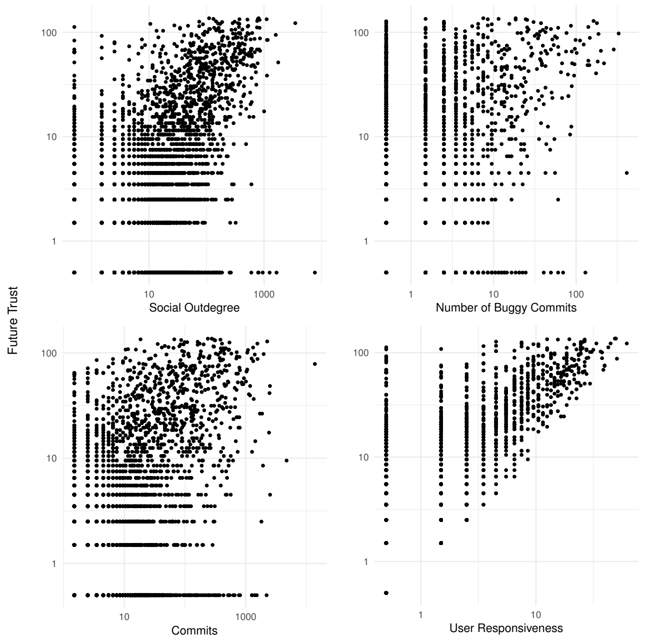

Figure 2: Future trust vs. selected attributes of visibility, expertise, productivity, and responsiveness. Axes log scaled.

Table 1: Trust model. User subscripts omitted; they refer to the developer under observation within the model.

	Dependent variable:			
	Future Trust (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 Trust 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

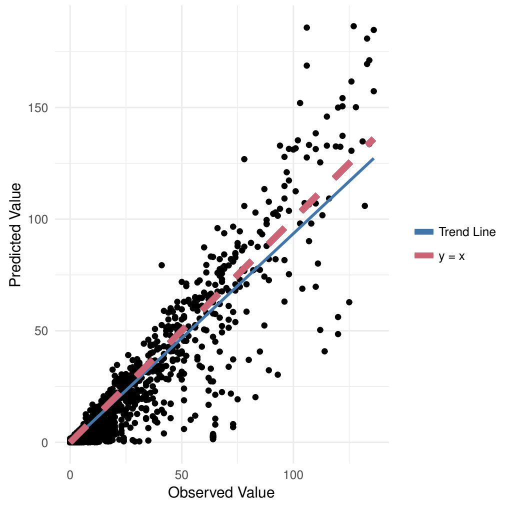

Figure 3: Predicted vs. observed values.

average and mean squared error are 0.910 and 15.769, respectively, indicating a good average 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 indicates that a higher social focus (in replying to others) and larger overall social outdegree leads to higher future trust - be it in the transition from zero to greater than zero trust, or in increasing trust. However, we see a negative coefficient for $\mathcal{ISS}_\kappa$, indicating that when others focus their calls on the observed individual, the observed's trust decreases[14]. This negative coefficient is not unexpected; as $\mathcal{ISS}_\kappa$ 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. For example, consider the case where there are 10 individuals that can call developer A. If all developers call developer 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, we note Posnett *et al.* [39] found that a higher value of $\mathcal{DAF}$ is associated with a lower raw cell count.

In sum, we see that having an overall larger social presence ($\mathcal{OSS}_\rho$, social outdegree) can increase one's future trust. As an individual, these values are much easier to increase than $\mathcal{ISS}_\kappa$, as $\mathcal{ISS}_\kappa$ is a function of indegree, and thus less in the individual's control. In addition, the positive effects of $\mathcal{OSS}_\rho$ and social outdegree can overtake the negative effect of $\mathcal{ISS}_\kappa$ if their sum contribution $0.103x_1 + 0.058x_2$ exceeds $0.283x_3$.

Expertise We see that the number of buggy commits a developer makes has a negative coefficient for the count component, indicating that a larger number of buggy commits leads to a decrease in trust. This is not unexpected, as one would believe that a higher expertise would lead to trust in the future. However, we see a positive coefficient for the zero component. This is puzzling at first, but may be explained as follows. It is known that contributions are

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

extremely important in order to gain trust [20], supported also by the large coefficient for commits in the zero component (0.453). As the number of bug commits is correlated with the number of overall commits by a developer, this positive coefficient indicates that contributing at all, regardless of whether or not your contribution is buggy, is important in getting the first call mention, and thus the first extension of trust.

Productivity For both the zero and count components, we see a positive coefficient for commits, indicating that increased productivity leads to higher trust. Of note is the fact that the coefficient for the zero model is very high - barely eclipsed by the size of $\mathcal{OSS}_\kappa$. This indicates that productivity is extremely important in receiving the first trust extension.

Responsiveness Interestingly and contrary to our hypothesis, for the count component, we see an insignificant coefficient, meaning that in the presence of all other variables, responsiveness is likely not important. 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 a trust value of 0.

Research Answer 1: *We see a net positive effect of visibility, as long as one's $\mathcal{ISS}_\kappa$ value is not large enough to counter the positive effects of $\mathcal{OSS}_\rho$ and social outdegree. We see that more buggy commits (a measure of negative expertise) leads to lower trust when one is already trusted, and higher trust if one is not already trusted, possibly explained by the idea that any productivity leads to a first trust extension. We see positive effects for productivity, and no significant effect of responsiveness.*

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 trust but low model predictions, and those who transition from zero to nonzero trust.

Sub-Case Study: High Observed Trust, Low Predicted Trust.

For this study, we manually examined those with less than 50 and greater than 15 observed future trust, nonzero observed trust, and a predicted trust of less than or equal to 1; *i.e.,* those along the bottom of the x-axis of Figure 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 trust is low.

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[15], a discussion about PRPL patterns[16], 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 [17]; 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 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. If we could capture an orthogonal metric of outside expertise, our model may better fit these individuals.

Sub-Case Study: Transitioning From Zero Trust.

For this study, we took a random sample of 10 individuals (out of 235) who had zero observed trust, but transitioned to nonzero trust 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. Thus, though all individuals have contributed to the project, there has not been a chance for trust extensions to be observed; those transitioning from zero trust to nonzero trust would likely have nonzero trust if the observation time split had been later in the project's age.

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 trust. 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 bugs in these commits (expertise), we see that the fruits of their labor have resulted in an extension of trust.

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

To answer this question, we require project-specific models of trust. Due to the sparseness of data, adding a factor to the existing model in Table 1 causes estimation to diverge. Thus, we fit simplified models with selected attributes of visibility ($\mathcal{OSS}_\rho$, $\mathcal{ISS}_\kappa$, social outdegree), expertise (bug 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 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. For the zero component, we use logistic regression; for the count component, we use Poisson regression.

[15]https://github.com/google/WebFundamentals/issues/4928

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

[17]The authors could not perform this in-depth study for some in this region as the project discussions were not in English.

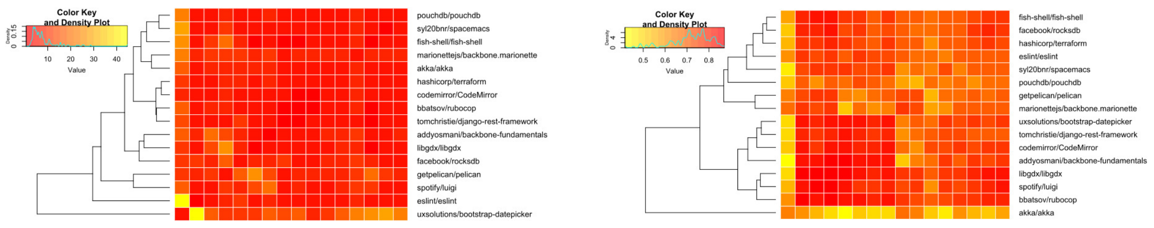

Figure 4: Cross-project predictive power heatmap for each project-specific model, count (left) and zero components.

Figure 4 contains heatmaps of the 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 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, illustrating hierarchical clusters of projects based on their predictive ability[18].

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 affecters of interest as compared to other projects. Figure 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 leads to a lower predicted trust value, 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 the zero component (though to a lesser extent).

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

[18]The heatmaps are symmetric.

RQ 3: Is there evidence of project-specific trust culture? Or are the affecters of trust a GitHub-wide phenomenon?

Figure 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 buggy commits for the zero component. However, we do see differences within columns, *e.g.*, for OSS_ρ in both model components, which is negative for some projects and positive for others.

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 trust culture differences on a per attribute basis. However, it appears there are things that don't change across projects, *e.g.*, the importance of commits in gaining more trust. In addition, the generally high cross-project predictive power shown in Figure 4 suggest 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.

Case Study: Project-Level Differences.

In Figure 5, we see heatmaps of coefficients for our project-level models. As a small case study, we seek to examine the question: why are some coefficients positive for a number of projects, and negative for others? As OSS_ρ seems to exhibit this behavior in both our count and zero models, and the coefficient is significant for our global model, we choose this variable for our study.

For the zero model, we see a negative coefficient for projects *uxsolutions/bootstrap-datepicker, pouchdb/pouchdb*, and *codemirror/CodeMirror*; this indicates that a higher level of specialization in one's replies leads to a lower future trust for these projects.

One explanation for this phenomenon could be due to a larger inner circle[19] as compared to other projects; *i.e.*, to gain trust one must become visible to 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, we see in both projects that 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

[19]Here, the inner circle is defined as the "top brass" of the project.

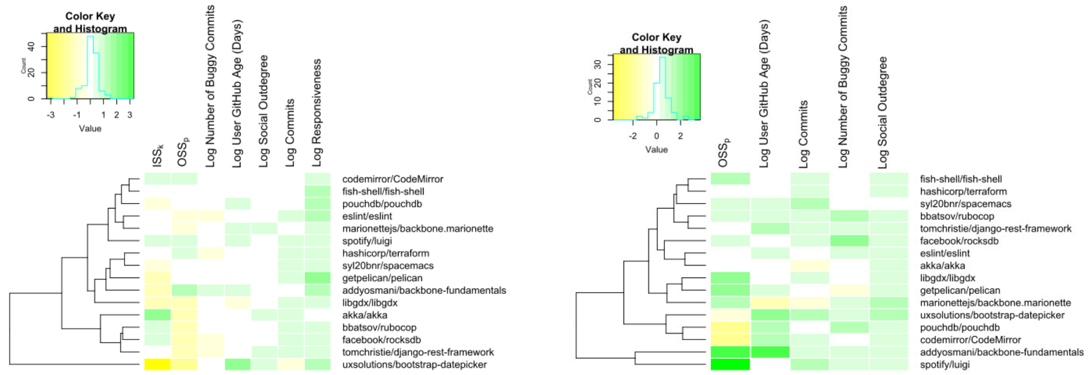

Figure 5: Heatmap of coefficients for project-specific models, (a) count and (b) zero components.

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 those projects with positive coefficients, we see a different behavior. In pull requests, it appears that the top project members are more open to calling on others to provide input. For example, 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 call on another contributor for their input, stating "*[I] would love to suggest your project to devs ...*". Recall that a positive coefficient for OSS_ρ indicates a specialization in reply behavior, indicating 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 must specialize in their social behavior towards the top contributors to become noticed; hence, more social specialization leads to a higher future trust.

Research Answer 3: *We see slight indications of project-specific trust culture as evidenced by varying coefficients for our project-specific models within the same affecter. However, generally high cross-project performance suggests that these differences may not matter much, especially in prediction.*

5 CONCLUSION

We performed a rounded quantitative study of the elements of trust in GitHub, as captured in calls to people during issue discussions. In this study large-scale data mining was supplemented with case studies on samples of discussions; both allowed us to triangulate our findings better. The well fitting, reasonable models, left us with the impression that the specific trust we chose to model can be modeled well from the data.

Some of our results were more obvious than others, *e.g.*, the effects of visibility and productivity on trust extension. Others were less so, *e.g.*, the non-effect of responsiveness and the positive effect of buggy commits on the initial extension of trust. From a security perspective, it may follow then, that trusting new people with the projects code is associated with more buggy code, perhaps via changes that they may introduce, which is certainly a concern. Based on these results, increased efforts towards training new people to the specifics of the project's code, *e.g.*, in creating a portal for newcomers [44], can be appropriate. Future work may include building online tools to facilitate transitioning people into the projects, *e.g.*, the creation of "trust profiles" which provide suggestions to new users regarding how to increase their trustworthiness, thus benefiting the project as a whole.

The idea that projects in an ecosystem have similar models of what it means to trust people 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 multi-dimensional nature of our trust outcome. It is also very reasonable that there would be cliques of projects in which the sense of trust is even more uniform than across the whole ecosystem, and our findings underscore that. Obvious open questions here are: how do models of trust get in sync in an ecosystem? And, to borrow from ecology, does the robustness of the trust models across GitHub convey any fitness benefit in this ecosystem? We can see a plausible mechanism that would offer an answer to the first: projects share people and people cross-pollinate the trust behavior across projects in which they participate. We leave the validation of this, and other models, to future work. The trust model robustness, likewise, implies some preference for success, be it by design or an emerging one, across the ecosystem. This can also be a function of people's mobility in the ecosystem and their preference for and vigilance to participate in more popular projects. Likewise, we leave the answers for future work.

5.1 Threats to Validity

There were challenges involved in all aspects of the work, largely due to the loaded meaning of trust. Many of those we anticipated and were careful to address. Once we settled on the idea of using @-mentions as signals, or extensions of socio-technical trust, we were able to connect our outcome with background theory of multiple components of trust. To define trust precisely, we necessarily had to narrow down our notion of trust to specific trust extended in issue discussion. Likewise for the multidimensionality of trust *vis-a-vis* the aggregate variables of trust.

We note that our case studies would benefit if done on a larger amount of data. However, the small 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.

REFERENCES

[1] Paul Allison. 2012. When Can You Safely Ignore Multicollinearity? https://statisticalhorizons.com/multicollinearity. (2012).

[2] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 137–143.

[3] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. 2007. Open borders? immigration in open source projects. In *The Fourth International Workshop on Mining Software Repositories*.

[4] Christian Bird, Nachiappan Nagappan, Harald Gall, Brendan Murphy, and Premkumar Devanbu. 2009. Putting it all together: Using socio-technical networks to predict failures. In *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on*. IEEE, 109–119.

[5] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. 2008. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 24–35.

[6] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. 2009. The promises and perils of mining git. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 1–10.

[7] Nico Blüthgen, Florian Menzel, and Nils Blüthgen. 2006. Measuring specialization in species interaction networks. *BMC ecology* 6, 1 (2006), 9.

[8] George G Brenkert. 1998. Trust, business and business ethics: an introduction. *Business Ethics Quarterly* 8, 2 (1998), 195–203.

[9] Joel Brockner. 1996. Understanding the interaction between procedural and distributive justice: The role of trust. (1996).

[10] Moira Burke, Cameron Marlow, and Thomas Lento. 2009. Feed me: motivating newcomer contribution in social network sites. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 945–954.

[11] Moira Burke, Cameron Marlow, and Thomas Lento. 2010. Social network activity and social well-being. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 1909–1912.

[12] A Colin Cameron and Pravin K Trivedi. 2013. *Regression analysis of count data*. Vol. 53. Cambridge university press.

[13] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. 2015. 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*. ACM, 817–828.

[14] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. 2013. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge.

[15] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. *First Monday* 10, 2 (2005).

[16] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 1277–1286.

[17] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368.

[18] Michael J Gallivan. 2001. 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 (2001), 277–304.

[19] Mohammad Gharehyazie, Daryl Posnett, and Vladimir Filkov. 2013. Social activities rival patch submission for prediction of developer initiation in oss projects. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 340–349.

[20] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, and Vladimir Filkov. 2015. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empirical Software Engineering* 20, 5 (2015), 1318–1353.

[21] Irving J Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika* 40, 3-4 (1953), 237–264.

[22] Charles Handy. 1995. Trust and the virtual organization. *Harvard business review* 73, 3 (1995), 40–51.

[23] James D Herbsleb. 2007. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*. IEEE Computer Society, 188–198.

[24] Eric von Hippel and Georg von Krogh. 2003. Open source software and the âĂIJprivate-collectiveâĂİ innovation model: Issues for organization science. *Organization science* 14, 2 (2003), 209–223.

[25] Liaquat Hossain and David Zhu. 2009. Social networks and coordination performance of distributed software development teams. *The Journal of High Technology Management Research* 20, 1 (2009), 52–61.

[26] Bryan W Husted. 1998. The ethical limits of trust in business relations. *Business Ethics Quarterly* 8, 2 (1998), 233–248.

[27] Ronald Inglehan. 1999. Trust, well-being and democracy. *Democracy and trust* (1999), 88.

[28] Sirkka L Jarvenpaa, Kathleen Knoll, and Dorothy E Leidner. 1998. Is anybody out there? Antecedents of trust in global virtual teams. *Journal of management information systems* 14, 4 (1998), 29–64.

[29] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 111–120.

[30] Thomas M Jones and Norman E Bowie. 1998. Moral hazards on the road to the âĂIJvirtualâĂİ corporation. *Business Ethics Quarterly* 8, 2 (1998), 273–292.

[31] 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 *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 574–585.

[32] Sunghun Kim, Thomas Zimmermann, Kai Pan, E James Jr, et al. 2006. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*. IEEE, 81–90.

[33] Roderick M Kramer and Tom R Tyler. 1996. *Trust in organizations: Frontiers of theory and research*. Sage.

[34] Nora McDonald and Sean Goggins. 2013. Performance and participation in open source software on github. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM, 139–144.

[35] D Harrison McKnight, Vivek Choudhury, and Charles Kacmar. 2002. Developing and validating trust measures for e-commerce: An integrative typology. *Information systems research* 13, 3 (2002), 334–359.

[36] Kenneth Newton. 2001. Trust, social capital, civil society, and democracy. *International Political Science Review* 22, 2 (2001), 201–214.

[37] Anne Oeldorf-Hirsch and S Shyam Sundar. 2015. Posting, commenting, and tagging: Effects of sharing news stories on Facebook. *Computers in Human Behavior* 44 (2015), 240–249.

[38] Michael O'Leary, Wanda Orlikowski, and JoAnne Yates. 2002. Distributed work over the centuries: Trust and control in the Hudson's Bay Company, 1670-1826. *Distributed work* (2002), 27–54.

[39] Daryl Posnett, Raissa D'Souza, Premkumar Devanbu, and Vladimir Filkov. 2013. Dual ecological measures of focus in software development. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 452–461.

[40] Lin Qiu, Han Lin, and Angela K-y Leung. 2013. 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 (2013), 106–121.

[41] Lionel P Robert, Alan R Denis, and Yu-Ting Caisy Hung. 2009. Individual swift trust and knowledge-based trust in face-to-face and virtual team members. *Journal of Management Information Systems* 26, 2 (2009), 241–279.

[42] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When do changes induce fixes?. In *ACM sigsoft software engineering notes*, Vol. 30. ACM, 1–5.

[43] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. 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*. ACM, 1379–1392.

[44] Igor Steinmacher, Tayana U. Conte, Christoph Treude, and Marco A. Gerosa. 2016. Overcoming Open Source Project Entry Barriers with a Portal for Newcomers. In *International Conference on Software Engineering*.

[45] Daniel Stewart. 2005. Social status in an open-source community. *American Sociological Review* 70, 5 (2005), 823–842.

[46] Eric Trist. 1981. The evolution of socio-technical systems. *Occasional paper* 2 (1981), 1981.

[47] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. 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*. ACM, 144–154.

[48] Quang H Vuong. 1989. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica: Journal of the Econometric Society* (1989), 307–333.

[49] Qi Xuan, Aaron Okano, Premkumar Devanbu, and Vladimir Filkov. 2014. Focus-shifting patterns of OSS developers and their congruence with call graphs. In

Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 401–412.

[50] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[51] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. 2014. Exploring the patterns of social behavior in GitHub. In *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*. ACM, 31–36.

[52] Yang Zhang, Huaimin Wang, Gang Yin, Tao Wang, and Yue Yu. 2015. Exploring the Use of@-mention to Assist Software Development in GitHub. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware*. ACM, 83–92.

[53] Yang Zhang, Huaimin Wang, Gang Yin, Tao Wang, and Yue Yu. 2017. Social media in GitHub: the role of @-mention in assisting software development. *Science China Information Sciences* 60, 3 (2017), 032102.

[54] Lynne G Zucker. 1986. Production of trust: Institutional sources of economic structure, 1840–1920. *Research in organizational behavior* (1986).