Developer Onboarding in GitHub: The Role of Prior Social Links and Language Experience

Casey Casalnuovo, Bogdan Vasilescu, Prem Devanbu, Vladimir Filkov
Computer Science Department, UC Davis, Davis, CA 95616, USA
{ccasal, vasilescu, ptdevanbu, vfilkov}@ucdavis.edu

ABSTRACT

The team aspects of software engineering have been a subject of great interest since early work by Fred Brooks and others: how well do people work together in teams? why do people join teams? what happens if teams are distributed? Recently, the emergence of project *ecosystems* such as GitHub have created an entirely new, higher level of organization. GitHub supports numerous teams; they share a common technical platform (for work activities) and a common social platform (via following, commenting, *etc*). We explore the GitHub evidence for socialization as a precursor to joining a project, and how the technical factors of past experience and social factors of past connections to team members of a project affect productivity both initially and in the long run. We find that migration in GitHub is strongly affected by pre-existing relationships; furthermore, we find that the presence of past social connections combined with prior experience in languages dominant in the project leads to higher productivity both initially and cumulatively. Interestingly, we also find that stronger social connections are associated with slightly less productivity initially, but slightly more productivity in the long run.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Programming Teams

General Terms

Human Factors

Keywords

GitHub, social aspects, onboarding, productivity

1. INTRODUCTION

While coding *per se* is a solitary activity, software development is very much a team sport: people organize into teams, spontaneously in open source, and by fiat in commercial software development. In open source the communication (computer-mediated) and collaboration activities (over version control) of software teams can be mined and studied. Prior research [3, 5, 7] has explored a great many aspects of how people join, work, and communicate within teams. Much of this research has been based on abundant data available from open source project repositories.

Teams, however, rarely exist in a vacuum; software teams interact with other teams in the same (large) organization, or within the same geographic area (such as Bangalore, or Silicon Valley) or those using the same technology platform (*e.g.*, Android) or competing in the same market (*e.g.*, trading platforms). These emerging "ecosystems" lead to new trans-project phenomena, such as migration of skills, technologies, and people between teams; the building of reputations; and the rise and decline of projects and individuals. These are clearly interesting and consequential phenomena; however the type of ecosystems listed above are difficult to observe and to study.

Recently, a new opportunity has arisen: the rise of "Web 2.0" software project ecosystems such as BitBucket and GitHub. These ecosystems provide technical platforms for social interaction, technical collaboration, and reputation building. They are exploding in popularity [18], with rapidly increasing numbers of users and hosted projects. Since there is a single technical platform for all the projects, it is now possible to mine an entire ecosystem, and observe important ecosystem-level phenomena. In this paper, we are specifically interested in the migration of developers between projects.

GitHub actually facilitates newcomers. The pull-based distributed software development process (as opposed to the earlier approach of a centralized repository with limited, controlled access) makes it a more democratic environment, where more people from different backgrounds can contribute. The common, shared technical platform, with single sign-on, as well as the extensive "social coding" features, facilitate the discovery of new projects, forging of new social links, and the opportunity to contribute in a range of ways.

In such a democratic, flat-land playground, do developers end up following their social links? How do those social links affect one's choices when wanting to join projects? Social capital in settings such as Twitter and Facebook are valuable, and can even be monetized. How do social connections and experience affect people's migrations between projects, and their technical contributions?

Starting from a dataset comprised of histories of co-participation in GitHub projects by prolific developers, in this paper we study the effect of developers' social links in the

GitHub ecosystem on their choices to join new projects, and their productivity once they do. In particular, we develop a measure of the strength of prior social links between a developer and existing members in a project, based on the number of developers in that project with whom the person has already co-participated in other projects. Using that, and statistical modeling, we find that:

- Developers preferentially join projects where they have prior social contacts. This affirms prior findings along the same lines done in other open source settings [13].
- Joining a new project in which there are some prior co-participants increases the developer's chances for initial contribution above baseline by 3.7% to 6.2%; similarly so for joining a project with dominant languages in which the developer has made prior contributions.
- Beyond the initial period, a developer's overall productivity in the project depends on their prior experience with the language of the project and their social connections in that project; having both gives a developer a great advantage in overall productivity, with an increase by 29.5% to 54.3% over baseline.
- However, developers who join environments where they have prior social connections but no prior language experience are 9.6% less likely to have cumulative productivity above baseline.
- As the strength of social connection increases, it is associated with a small decrease in productivity in the initial period, perhaps associated with coordination overhead between team members. However, stronger social links indicate higher cumulative productivity.

The rest of the paper is organized as follows. The theory behind our research and the research questions are in Section 2, followed by the methodology in Section 3. The results and discussion are in Section 4, and threats to validity and conclusion in Sections 5 and 6, respectively.

2. THEORY

2.1 Team Formation in OSS

Compared to organized, corporate project teams ("fiat teams"), open source software (OSS) teams tend to be quite fluid and diverse. Typically, OSS teams consist of a mix of professional developers and volunteers, often geographically distributed, with varied demographic features, personalities, and expertise. Moreover, OSS teams tend to be self-organized [6], and they are more fluid than their fiat counterparts. In OSS, teams form and dissolve organically around each task at hand, with contributors that come and go as they please (especially the volunteers), resulting in high turnover [30]. GitHub OSS teams are no different [36].

Prior work on team formation, be it self-organized or fiat, indicates that it is a deliberate, strategic process, in which individuals attempt to satisfy personal and group objectives [27]. In self-organized teams, the process is typically bi-directional, in that individuals seek to join groups that will enable them to satisfy personal needs, and groups seek members that can fit in and will contribute to reaching the groups' objectives [21]. In OSS, *e.g.*, projects need contributors with particular skills, while developers who are looking to join projects are driven by a direct need for the software, enjoyment of the work itself, learning opportunities, or reputation building [19, 20].

However, it is difficult for OSS developers to predict before joining the extent to which prospective target projects will be successful and will enable them to maximize the realization of their goals [13]. Instead, to deal with uncertainty, developers will rely on other cues when forming impressions about other developers and projects. Research on self organizing networks suggests that people prefer (i) repeated collaborations (cohesion) over new ones, to benefit from prior interactions, greater mutual trust, and increased knowledge about each other's technical and social skills, and (ii) collaborations with established (higher status) actors, which increases their own motivation and their perception of the likelihood of project success [12, 35]. Teams comprised of members with prior joint experience can coordinate their efforts and leverage each other's expertise more effectively [9, 26]. Over time, the teams will engage in more meaningful interactions, adjust to the surface-level differences between them, and benefit from their different cognitive frameworks and value sets (*i.e.*, their information-processing [31] approaches), thus improving the team's efficiency and decision making processes [14].

In OSS, the importance of socialization and social ties for team formation are well documented. Research suggests that developers use social ties created during joint participation in past projects when choosing new collaborators. Specifically, prior work on SourceForge [13] has found that developers are more likely to join projects that are initiated by others with which they have interacted intensely in the past, but collaborative tie strength with other team members does not seem to have a significant effect on their choice to join those projects. We find support for preferentially choosing projects on GitHub, but we also study how the strength of social ties influences one's initial and cumulative productivity after joining, rather than just the choice of joining.

However, joining an OSS project is itself a complex process [32], influenced by a multitude of social and technical factors. There are also different roles through which a newcomer can advance in a typical OSS project based on their level of commitment, commonly referred to as the "onion model" [44]. Typically, one starts from the periphery by contributing bug fixes, documentation, small feature improvements, and participating in discussions on mailing lists, and tries to advance through the ranks towards the core, where they are granted direct access to the project's source code repository. Prior work suggests that a congruence of the newcomer's social and technical activities is essential for successful onboarding in OSS [3, 4, 8, 10, 17, 40]. Sustained, high quality contributions containing working and well-tested code are necessary to add technical value to the project, and help other project members increase their trust in the developer's ability to contribute. Similarly, good communication and social skills signal that the developer can integrate well with other team members, and can help her advance through the ranks. The social and technical activities of newcomers, *i.e.*, who they talk to, how many social links they develop with other project members during the onboarding period, how well they communicate, and how active they are at submitting patches and identifying and fixing bugs, all contribute to increase one's chances of advancing through the ranks.

Still, not all team formation attempts are successful. Various technical and social contribution barriers, including

expectation breakdowns, reception problems, insufficient community support, and steep learning curves, can impede newcomers from completing the onboarding process [32, 33]. The social environment of the project also plays a role in team formation in OSS, affecting both the rate at which developers join a project as well as the chance that a newcomer will become a long-term contributor [45, 46].

2.1.1 Developer Migration in OSS

A special case of team formation in OSS is developer migration between (interconnected) projects part of a larger software ecosystem, *e.g.*, Gnome, Apache, GitHub. Despite the abundance of literature on onboarding of newcomers in individual OSS projects, very few empirical studies of migration of developers *across* projects exist. Weiss *et al.* [41, 42] studied mailing list email exchanges between contributors to Apache projects, and uncovered developer migration patterns within the ecosystem. They found that migration of developers between different Apache projects is a common occurrence (*i.e.*, in their data most newly started projects included at least one large group of developers that migrated from another Apache project), and that the strength of social ties influences the migration behavior, with evidence for preferential attachment (*i.e.*, while many developers will migrate in small groups, some well-connected developers will move in large groups). Recently, Mens *et al.* [24, 25] have began studying the migration patterns of Gnome contributors, from the perspective of the ecosystem's survivability. In this context, a sudden loss (or intake) of contributors to projects part of the Gnome ecosystem may be indicative of an important environmental disturbance, therefore analyzing effects associated with the intake, retention and loss of developers at the level of individual Gnome projects may provide insights into how the ecosystem returned to an equilibrium after this disturbance.

2.2 Social Coding in GitHub

The propensity of people to form new teams, or migrate to existing ones, is greatly amplified by technical standards and platforms. GitHub certainly provides a common distributed code sharing and versioning platform. The widespread familiarity of developers with systems like `git`, and the flexible collaborative processes that it supports, certainly facilitates contributing to, starting up, and migrating to team projects in GitHub. However, GitHub does much, much more to facilitate teams and migration, through the *social coding features* it offers. These features allow developers to track each others' activities, and thus form detailed impressions of their social and technical abilities and behavior.

The activities of developers in GitHub (*e.g.*, reporting bugs, submitting pull requests, commenting) are all not merely recorded; they can also be "followed" by others, and readily made available on user-identified "home" pages. These records provide a valuable perspective on not only the technical abilities of a person, but also their social skills and proclivities. Lima *et al.* [22] build networks of these followers and also collaborators based on push events of commits on GitHub. They find that following relations are often one directional and that small teams are often comprised of geographically close individuals. Marlow *et al.* [23] report that these GitHub records play a key role in how developers form impressions of each other, based on evaluations of the work products contributed by others. Brian Doll, in an interview by Storey [1], claims that "the number one way of getting a job...right now" is to showcase one's work on GitHub; his pitch for GitHub is supported by an article in CNET [34]. Dabbish *et al.* [7] report that developers indeed use these records not only to form impressions of others, but also manage their own on-line reputations.

These findings suggest that team formation and inter-team migration are strongly influenced by perceptions formed through GitHub records; as discussed below, these concepts inform our research.

2.3 Research Questions

Given that the social aspects of GitHub and the pull request mechanism lower the barriers to entry for newcomers, allowing developers to more easily participate in a number of projects simultaneously, we wish to understand how developer migration manifests itself in this social environment. We begin with a question to understand if developers preferentially select to join projects that have on board people with whom they have prior ties:

RQ1: Do prior social connections matter when developers join new projects? That is, are they more likely to join projects in which there are developers they have already collaborated with in the past?

When a developer joins a project, there are many different factors that can influence their productivity. Developers' activity on GitHub will generally rise and fall throughout their tenure, and not stay constant. If they are involved in multiple other projects, their existing commitments might lower the levels of contribution in the new project, simply because they need to divide their attention between many different venues.

Additionally, the theories reviewed above suggest that people are aware of each other's technical skills, and teaming up and collaboration will be influenced by those attributes. Thus, if a project involves languages with which a developer has no prior experience, they may not be as productive initially in the project. Moreover, a developer's initial productivity in a new project may vary depending on their having significant social connection to the project, based on prior social links with other developers. We draw from the theory on teams that social connections are key for future collaborations. Putting all these into a broader context of an environment with observables we can control for, we ask:

RQ2: How does the presence of past social connections, and their strength, influence initial developer productivity in both familiar (past language experience) and unfamiliar project environments (no past language experience)?

Finally, some of those effects may be moderated by time. As a developer's familiarity with the project changes, and they become more invested in it, so will their attitude towards contributing in that project. Naturally, projects also vary in popularity over time, and people change focus, as interests change and social connections develop. Focusing on the overall productivity of a developer throughout their tenure with a project, we ask:

RQ3: What is the effect of past experience and prior social connections on how productive a developer will be overall in a project?

3. METHODS

3.1 Data Gathering

Although GitHub is presented as a platform for "social coding", it is well known that it has numerous inactive or personal projects, which are disconnected from meaningful team-based software development [18]. Furthermore, most GitHub users do not have a sufficiently long record of commits to draw meaningful conclusions about their project-joining behavior. Therefore, our study focused on prolific developers who contributed to multiple projects. Using data from the GHTorrent [11] dump dated 2014/11/1 and our previously assembled diversity data set [37], we identified an initial set of 1,274 developers with a long GitHub contribution history. We selected developers active for at least 5 years, and who contributed at least 500 commits to at least 10 different repositories (*i.e.*, projects). We exclude forks as recommended when mining GitHub [18].

Between them, these developers contributed to 65,280 different projects. To obtain more detailed project data (*e.g.*, on the size and contents of each code change) than offered by GHTorrent's fast MySQL database, we extract data directly from each project's commit logs. We attempted to clone the main repository for all 65,280 projects in order to obtain their `git` logs, and succeeded in 58,170 cases. In cases where the cloning failed the project no longer existed on GitHub (their project pages returned "404 Not Found" errors) when we collected data in November 2014.

From here, we textually parsed all `git` logs to identify the files touched in each commit, and what lines were added and deleted. We retained the full file paths and extensions of all the files. There were 78 projects that raised an error when parsing their log files, and were subsequently excluded. In total, we collected detailed data on 1,255 prolific developers who together contributed to 58,092 projects.

3.2 User Aliases

When assigning authorship to a commit, we examined only the author fields contained in the `git` logs. These fields identify the developer who is the original author of the code, and not the committer, who is the person submitting the commit to the main repository (and may be different from the author).

Unfortunately, developers may use multiple emails and user names in their commits; without accounting for possible aliases, this can lead to contributions from the same individual being associated with multiple identities. To address this challenge, we reuse prior results on identity merging from our GitHub diversity data set [38], wherein we mapped aliases belonging to the same person to a single entity. The details of these alias resolution techniques are discussed *op. cit.*

3.3 Temporal Modeling

Modeling temporal information is complex and can be challenging to interpret correctly; we adopt a simplified approach. We approximate developers' behavior over time with a sequence of 11 fixed-size time windows. We choose the beginning of 2009 as the starting point, since GitHub officially launched on April 10, 2008.[1] The first time window, labelled t_0, is different from the other 10. It represents the cumulative behavior of the developers until the start of 2009, and aggregates the initial habits of developers. The rest of the time windows, labelled t_1 to t_{10}, measure six month intervals from the start of 2009 to the end of 2013. That is, t_1 measures January 1, 2009 to June 30, 2009, t_2 measures July 1, 2009 to December, 31, 2009, and so on until December 31, 2013. Each window aggregates the project joining behavior of the developers during those 6 months.

Finally, when examining the productivity of developers, we chose projects that were joined between t_1 to t_9, and not prior or hence. This guarantees at least 1 year of commit history for all projects in our productivity studies.

3.4 Language Estimation

We classified each file that had been committed to into one of 34 language categories. These categories included 31 programing languages and 3 special categories. The 31 programming languages were chosen from the combination of the top 20 languages on GitHub and the languages that appear by default in GitHub drop down menus. The three special categories included an *other* category for files in less popular languages and non code files, an *ambiguous* category for files which could not be classified correctly, and a *special* category for *C/C++*. This last category is specifically for .h files in projects that used both *C* and *C++*.

To determine the programming language in which a file is written, we used a mapping from file extensions to programming languages provided by the GitHub linguist project.[2] GitHub linguist itself works more precisely than that, incorporating file contents with extensions to help identify the language, but our log file (vs source) parsing limited us to using only file extensions. As some file extensions are shared between different types of files, we had to remove ambiguity using additional contextual information. For instance, we used the project that an ambiguous file was from to narrow down what language it belonged to. Each project on GitHub has a label for the project's main language, so if we had an ambiguous file with an extension that could be used by this language, we classified it as such. For instance, `.m` files can be *Matlab* or *Objective-C* files. However, if the repository is labelled as *Objective-C*, we classify this file as such. Additionally, we use the context provided from other files in the project to resolve ambiguous cases. If a file type could belong to a set of languages, we examine the classification of the other files in the project. If exactly one of the languages in the set of possibilities exists within the project, all files in this ambiguous set are classified as written in this language. This method is conservative: we would rather ignore an ambiguous file than incorrectly classify it. Of all files in all projects, only about 1% remained classified as ambiguous.

3.5 Statistical Modeling

We perform statistical modeling at two levels of granularity. First, we address **RQ1** using a cumulative characterization of project joining behavior, combining all possible determinants of one's preferential project joining behavior (*e.g.*, project size, popularity, etc.) into a single choice. Second, we perform a more fine-grained regression modeling of developers' productivity in newly joined projects, taking into account their past technical experience and strength of social links with other project members, in order to answer **RQ2** and **RQ3**.

[1] `https://github.com/blog/40-we-launched`

[2] `https://github.com/github/linguist`

3.5.1 Modeling Project Joining

From the perspective of a developer looking to contribute to a new project, one can choose between starting a new project and joining an existing one. Among prospective projects, one can choose between those with prior social connections and those without (we exclude all projects that the developer starts, as those necessarily lack any prior connections). We say that two developers have a social connection if they both have committed to at least one same project in the past. Now, a developer has a prior social connection *to a project* if there is a social connection between the developer and a member of the project who has joined at least one day before.

Since we have temporal data, we test joining preferences in two ways. First, we test the hypothesis that cumulatively, over all time windows of observation, a developer chooses to join projects randomly, irrespective of prior social connections in them. That is, in aggregate over all time windows, a developer chooses to join an existing project at random. The alternative, then, is that the developer preferentially joins projects with prior social links.

We also look at project joining in a stricter setting: we test whether a developer prefers to join projects non-randomly in each of the 10 time periods (*i.e.*, is random joining, the null hypothesis, rejected for each interval?). In this case, for each developer we exclude time periods where they either did not join any projects or did not yet have any prior social links established.

To test the above null hypotheses, we use the hypergeometric distribution [28]. Let N be the number of projects that a developer has not yet joined by the end of our observation period (*i.e.*, over all time periods between 2009 and 2013), K be the number of projects among them where the developer has prior social connections, n be the number of projects that the developer (eventually) joins, and, finally, k be the number of projects the developer joined where they have prior social links. In general, given a set of size N comprised of two distinct classes of objects, say SUCCESS and FAILURE, the hypergeometric distribution models the sampling without replacement of $n < N$ objects from this set. Then, having K SUCCESS-es among the n samples has a well defined probability [28]. So, when in our data we observe a total of k successes, or projects joined with prior social links, we can tell exactly how far that is from random [29]. We choose the standard p-value < 0.05 as the statistical significance cutoff for rejecting the null hypothesis for each developer. Thus, if we reject a null hypothesis there is only a 5% probability it was a false positive. Since we are testing thousands of null hypotheses and calculating how many of them were rejected, we use a family-wise adjustment to the p-values to correct for false positives. We choose the robust *Benjamini-Hochberg* correction [2] for this.

3.5.2 Modeling Productivity After Joining

To model the effects of a developer's past technical experience and strength of social links on their productivity in a project, we use negative binomial regression [28]. This regression is a generalized linear regression model effective in handling over-dispersed count data (with its counterpart the quasi-poisson regression, they are used interchangeably in this situation). Over-dispersed counts are those where their variance is much larger than their mean [39]. Our response counts are over-dispersed. We present the results of the modeling as odds ratios, *i.e.*, as the *exp* (exponent with base e) of the regression coefficients. They correspond to the multipliers compared to a baseline (null) behavior.

We use the fraction of total deviance explained ($1 - residual\_deviance/null\_deviance$) as an assessment of the goodness of model fit, *i.e.*, McFaden's pseudo-R^2 [28]. To understand collinearity among the predictors we use the VIF [28], or variance inflation factor. In all our models, the variance inflation factors were below 2, except for those between the interaction terms and their comprising factors, which is expected. The low VIF's of our model coefficients imply that collinearity between them is not an issue. We describe the regression variables next.

Response Variable

Productivity.

We measure the productivity of developers as the number of file changes made to the popular source code language files mentioned above. For example, if a developer authored 3 commits in a time period and touched 4 *python* files in the first commit, 2 text files in the second, and 5 *java* files in the third, we say that the developer has made 9 file changes to source code files. We apply this measure to the individual time periods, where we consider only file changes within the given period, and to the cumulative measure of total number of file changes a developer has made to a project.

However, we want to discuss a few other metrics of productivity that we considered using but chose not to. We considered using both the total lines added in the commits and the sum of lines added and deleted as a measure of productivity, as well as simply the number of commits. We chose file changes over commits as they give us a slightly finer granularity of the work done in the projects. As for measuring productivity by changes in lines of code, we had originally built our models using these metrics. However, upon careful examination of the data, we believe that noise in `git` line change data makes it a less reliable measure than file changes. Upon joining a project, developers may merge code from elsewhere, which will distort line change data far more than file change data. Nevertheless, we compared models for joining behavior in the first time period for file changes and with lines added and removed, and we found that the signs and significance of the coefficients were *consistent* across these models.

To avoid effects of outlying behavior, we ignored developer project pairs where the first or the second time period had more than 50 file changes. Large commits may indicate a different type of activity than usual developer behavior [15]. We considered models of the first time period, without subsetting the data. The signs and significance of the model coefficients did not change, compared to our models reported here, but the outlying large commits have high leverage and artificially inflate the amount of variance our models explain. For the same reasons, we filter out very high values for the strength of prior links (defined next), *i.e.*, those higher than 20. After applying these filters, we retained 39,461 samples of an original 51,593 samples of project joining behavior. Finally, in the cumulative models, we further subset the data to avoid projects where developers had made over 500 cumulative file changes. This only removed an additional 210 projects from the data (out of the 58,092 we started with).

Independent Variables

Prior Social Connections.

As developers join new projects, they are exposed to other project contributors. Developers that appear together in many projects may develop friendships or working relationships. These relationships of developers repeated across projects may affect developers' choices of projects to join and behavior within these projects. To find such putative social connections, we calculated their precursors: the set of stable pairs of developers among our projects. We define a stable pair as two developers who appear together in at least two different projects, based on the author fields of the project's commits. This set is an overestimate of actual social connections built on the above mechanism; it should certainly not be considered a strong measure of collaboration, which is better defined as developers touching similar files within a short time frame [43]. Lima *et al.* have built a similar stable developer-pair network, though they used push events on GitHub rather than the commits themselves [22].

Naturally, the social connections between developers are not all of equal strength. Therefore, we incorporated several other factors to weight these ties more appropriately. First, developers who work together in many projects are likely to have stronger ties than those together in only few. Second, the strength of the social connection can change over time, as two developers may work on an increasing or decreasing set of shared projects. Finally, developers with similar interests may contribute to many of the same projects, without necessarily developing a strong relationship. This is more likely in large projects with many other developers participating, so the team size in each of the developer's shared projects should be taken into account. Putting it all together, *the strength of a developer's social connection to a project is the aggregate of their social connections to all developers already in the project that they have worked with previously.*

To precisely measure the strength of a developer's social connection to a project, we define some variables first. Let D stand for a developer, p for a project they join, and t_j for the time period of joining. Let also, p_f be the number of people in p with whom D has worked together elsewhere, prior to t_j. And for each person i let $K_{i,t_{j-1}}$ be the number of projects in which D and i worked together, before t_j. For each of these $K_{i,t_{j-1}}$ projects, let $a_{i,k,t_{j-1}}$ be the number of authors in the project prior to t_j. Then, we calculate D's initial social connection to p with the following formula:

$$\sum_{i=1}^{p_f} \sum_{k=1}^{K_{i,t_{j-1}}} \frac{1}{a_{i,k,t_{j-1}}}$$

In other words, the strength of a social connection to any one developer is the sum of all their past shared projects, normalized by the number of authors in the project prior to the time of joining. Figure 1 illustrates our metric's variability as a function of the count of past connections. This highlights how even if a developer has worked with only a couple of members of a project they are joining, they may share many past projects and may have a stronger connection to this project than to a project where they have only small incidental connections to many developers. We also experimented with several other measures of social connection. Those included removing the team size normalization, selecting only the weight of the strongest tie to the project, and also mapping social connections only when both developers were major contributors to past projects. However, we found the regression models with these metrics similar, but with less significance or explanatory power. Finally, the team size of the current project can also play a role, but we control for that variable separately in our regressions.

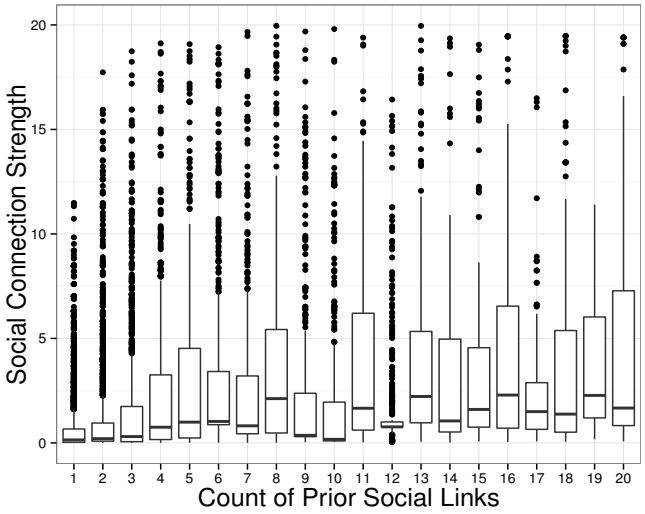

Figure 1: **Comparison of our metric of social connection strength with the count of developers on a project team that a developer has previously worked with (*i.e.*, p_f). The data has been cut off at 20 for each metric to remove outliers.**

Past Experience.

A developer's past experience with the languages used in a project is the count of file changes they have made in these languages to any project prior to joining the new project. To illustrate this metric, consider a project that uses *C++* and *python* code. Suppose one developer joins and contributes only *C++* code. Then, this developer's relevant past experience (after this project) consists only of file changes in *C++*. However, if another developer contributes in *python* and *C++*, then we would add changes in both languages as relevant past experience.

Control Variables

Many additional factors can influence how productive a developer may be in a project. To better isolate the effects of social links and past experience, we included as controls in our models the following: project team size, a developer's overall productivity during their tenure with the project, whether they were a project founder, and the time period when they joined.

Project Team Size.

To control for the size of the project, we count how many people were already "affiliated" with the project at the time of joining. For an individual to be considered affiliated with a project, they must have authored at least one commit recorded in the project's main repository.

Total Productivity Per Time Period.

A developer's productivity in any one project will depend on her total activity for the time period. Thus, we include two controls to adjust for developer productivity in a time period. These are: 1) the total number of file changes the developer

makes in the time period; and 2) the total number of projects the developer has contributed to in this time period. In the cumulative model, there is no obvious direct analogue to the number of other projects contributed to in the time period. Therefore, we substitute this measure with the count of past projects the developer had joined before joining this project. In the model studying initial productivity, we do not include both prior projects and currently active projects as they correlate highly with each other.

Joining and Founding.

One important distinction to make when studying a developer's joining behavior is whether they are joining projects or if they are starting a new project. Obviously, a developer who starts a project alone will not have any connection to team members when joining as they are the first to join. Therefore, we must distinguish between these two types of projects for each developer. We say that a developer has founded a project if they have committed to this project on the first day that commits appear in its main repository. Otherwise, they are considered as joining the project. One concern with this definition is that if an existing project is imported to git, the commits of all the existing developers could appear on the first day. We checked the number of first day committers for each project and found only 16 projects with 10 or more committers on the first day, with the largest having 31 first day committers. As 51,615 of the projects had a unique founder, we do not believe this will affect our results significantly.

Time Period Joined.

When considering the cumulative model of productivity over the entire project lifespan, the time periods in which developers begin contributing will affect how much work they can accomplish. A developer who joins a project in 2009 will have the opportunity to contribute more than a developer who joins a project in 2013, relative to a specific later data collection time.

Other Controls Considered.

We considered using other metrics for project size besides the number of project members, but they all correlated with team size and raised the VIF scores of our models beyond acceptable ranges. Team size was found to have the greatest effect, so we chose it as our measure.

4. RESULTS AND DISCUSSION

4.1 Social Links and Joining a Project

To assess if developers preferentially join projects where they have prior social connections, or links, we split all projects joined by a developer into two groups: those with and those without prior social links for that developer. As detailed in Section 3.5.1, we first tested the null hypothesis that cumulatively, over all time periods between 2009 and 2013, a developer joined projects with past social links as expected by chance. The first row of Table 1 shows the number of null hypotheses tested (one per developer) that were rejected, or not, at 0.05 significance level, after correcting for multiple hypothesis testing 3.5.1. We see that just over 90% of our prolific developers preferentially join projects with past social connections. Next, we tested at finer resolution, to see if the developers' joining preference in each time period is non-random. We found, see second row in Table 1, that in just under 60% of the time periods that is the case. Finally, in the third row we aggregate for each developer their per-period results. We find that 56.7% of them preferentially join projects with past social connections in more than half of the time periods.

Table 1: **Tests of null hypotheses: "developers randomly join new projects". The "Rejected" column indicates preference for projects with past social connections. "Cumulative" means over all time periods.**

Description	#Rejected	#Not Rej.
Cumulative		
Num. Hypotheses (Num. Developers)	1081	119
Per Period		
Num. Hypotheses (Periods)	4199	2854
Median of Periods		
Num. Developers with > half periods non-random	680	520

These are consistent with a positive answer to RQ1.

While precise, the above approach is coarse, and combines all possible determinants of one's preferential project joining behavior, *e.g.*, project size, popularity, etc., into a single choice. Such an approach works for our goal of motivating our two latter, more substantial research questions. It is also in line with similar prior results. In a comprehensive, but narrower study, Hahn *et al.* [13] showed that prior social links with project initiators do matter for developer migration to new projects, when controlled for a number of factors, like project and team based observables. Due to the data mining nature of our study, we cannot distinguish between incidental and purposeful relationships.

4.2 Developers' Initial Productivity

Here we examine the initial productivity of a developer after they join a new project. After pre-processing and filtering[3] our data set consists of 1,234 distinct developers, who have joined 22,028 distinct projects. We measure initial productivity in terms of the number of file changes to source code files within the first 6-month period after joining a project. Then, we model this productivity as a function of a number of control variables, as described in methods, and two independent variables: (a) the strength of existing social links to prior co-developers that the joining developer finds within the new project, and (b) her familiarity with the dominant languages in the new project.

Since we use a complex measure for prior social link strength, we recognize that having any social links (non-zero friends) in a new project may be unrelated to the strength of those links to the developer. Likewise for prior experience with a project's dominant language. Thus, we capture the dependent variables in two ways: continuously and categorically (or nominally). The continuous variables are *prior_link_strength* and *prior_exp_strength*.

[3]Recall that our data set includes 58,092 projects and 1,255 users; we consider their project joining behaviors in time periods t_1 to t_9. We filter out contribution of large sizes in the first or second time periods, those with large outlying prior social links, and look at popular source code languages only. See the methods section for more on this setup.

Table 2: **Odds ratio coefficients from a negative binomial model for the number of initial file commits to a project. [16]**

	# initial_file_changes
Dependent Variable:	Odds ratios
has_prior_links	1.127^{***}
	(0.032)
has_prior_exp	1.101^{***}
	(0.023)
prior_link_strength	0.980^{***}
	(0.002)
prior_exp_strength	1.000^{***}
	(0.00000)
# projects_in_initial_period	0.998^{***}
	(0.0001)
# total_file_changes_initial	1.000^{***}
	(0.00000)
authors_prior_to_join	1.000^{**}
	(0.00000)
is_founder	2.573^{***}
	(0.013)
time_period_joined	1.002
	(0.002)
has_prior_exp: has_prior_links	0.942
	(0.033)
Intercept	5.233^{***}
	(0.023)
Observations	39,461
Log Likelihood	−117,086.200
θ	1.120^{***} (0.009)
Null deviance:	50739
Residual deviance:	40624
Akaike Inf. Crit.	234,194.500

Note: $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$

For their categorical counterparts, we use two dichotomous variables. The first one is *has_prior_exp*, defined as TRUE (coded as 1) when *prior_exp_strength* (in the programming languages dominant in the project) is greater than 0, and FALSE (coded as 0) otherwise. The second is *has_prior_links*, defined as TRUE (coded as 1) when *prior_link_strength* is greater than 0, and FALSE (coded as 0) otherwise. In addition, to deconvolve a possible relationship between the categorical variables, we added an interaction term $(has\_prior\_exp == 1) * (has\_prior\_links == 1)$.

We use a negative binomial multiple regression for our model, with a log link function [28]. Table 2 shows the odds ratios and significance for the model. The fraction of total deviance explained by the model is 19.9%.

Of the controls, all but the time period of joining are significant, although most have negligible effects. Being a founder of a project is the notable and largest predictor of one's initial productivity, which is not unexpected. The odds ratio of 2.573 means that a developer who is joining a new project is 2.573 times likelier to have higher productivity in the initial period if she is a founder of that project. There is also a very slight slow-down associated with working on more projects during the initial time period after joining (*#projects_in_initial_period*).

Next we look at the categorical independent variables. The interaction effect tells by how much the effect of prior familiarity with the project's dominant languages differs between those developers who have and those that do not have prior social connections to team members in the project they are joining. But, it does so in multiplicative terms. To understand the individual terms in the presence of the interaction, we follow the standard practice of varying one while holding the other constant, and vice versa.

The odds ratio for *has_prior_exp* is 1.101, which means that the odds of initial changes are 1.101 times, or 10.1% higher for developers with prior experience in that project's languages. Since we have added an interaction term between *has_prior_exp* and *has_prior_links*, the above effect of having prior experience refers to developers without prior social links in the project they are joining. The effect of prior experience for developers with prior social links is 0.942 times the effect for developers without prior social links, $0.942 * 1.101 = 1.037$, or 3.7% higher odds.

Similarly, the odds ratio for *has_prior_links* is 1.127, which means that the odds of initial changes are 1.127 times, or 12.7%, higher for developers with prior social links in that project. Again, because of the interaction term, this effect is for developers without prior experience in the languages of the project they are joining. The effect of prior social links for developers with prior project's language experience is 0.942 times the effect for developers without prior social links, $0.942 * 1.127 = 1.062$, or 6.2% higher odds.

The results also show that the interaction term in this model, although sizable, is not significant. Thus, the two dichotomous variables could have been modeled separately, with the same fidelity. In fact, a model as above but without the interaction term returns odds ratios for *has_prior_links* and *has_prior_exp* of 1.062 and 1.037, respectively.

Finally, we look at the continuous dependent variables. Interestingly, the social link strength odds ratio is smaller than 1, thus lowering the odds of baseline commit contributions by a factor of 0.980, or close to a 2% reduction for every unit change of *prior_link_strength*. The amount of prior experience in the project's dominant languages doesn't have a sizeable effect on productivity.

In summary, the answer RQ2 is that having either prior experience with the dominant languages in the new project or prior social links in it will increase the odds between 3.7% and 6.2%, for additional initial file changes over baseline by the newcomer, with social links doing slightly better than prior experience. However, joining a projects having stronger social links decreases the odds of being more productive *in the initial period* than the baseline by 2% for each unit of *prior_link_strength*.

4.3 Developers' Overall Productivity

Next, we look at the developer's productivity aggregated from the moment they joined a new project until the end of data collection, *i.e.*, their overall productivity in our data

set. This data set is a subset of the prior one, where we also remove developers with very large cumulative contributions to projects. This leaves us with 1,234 developers and 21,965 distinct projects. We use a negative binomial multiple regression, with a log link function [28]. Our model here is similar to the one above, except we add an additional independent variable: the initial developer's productivity, *i.e.*, the dependent variable from the previous model. Table 3 shows the effects of our independent and control variables. In comparison to previous model, the fraction of deviance explained in the cumulative model is 34.7%.

The model coefficients vary quite a bit with respect to the initial productivity model. The controls are similarly uneventful except for the expected positive effect of being a project founder. However, in the cumulative model it is lower than before, with an odds ratio 1.630. This is consistent with founders having a sustained higher productivity long-term, although lower than in the initial, likely euphoric, period. Also, the time period in which a developer joins is important here: the later one joins, the lower their overall productivity. This is expected, as these developers have been contributing for a shorter time. Finally, the size of the contributions in the initial period ($\#initial\_file\_changes$) correlates with an increase in cumulative contributions by 5.9%. As might be expected, greater productivity initially makes a higher total contribution more likely.

The categorical variables here also tell an interesting story, and act differently than in RQ2. Unlike previously, in this model the interaction term is sizeable and significant. The odds ratio for *has_prior_exp* is 1.077, which means that the odds of overall file changes are 7.7% higher for developers with prior experience in that project's language. Due to the interaction term between *has_prior_exp* and *has_prior_links*, the above measure of productivity is for developers without prior social links in the project they are joining. The effect of prior experience for developers with prior social links in the new project is 1.433 times the effect for developers without social links, or $1.433 * 1.077 = 1.543$, *i.e.*, the odds are 54.3% higher for above baseline overall contributions. Thus, in the long run experience matters, though its positive effect on productivity is stronger in the presence of past social connections.

On the other hand, the odds ratio for *has_prior_links* is 0.904, or 9.6% lower, for developers with prior social links in that project. Again, because of the interaction term, this effect is for developers without prior experience in the languages of the project they are joining. The effect of prior social links for developers with prior language experience is 1.433 times the effect for developers without prior experience, $1.433 * 0.904 = 1.295$, or 29.5% higher odds for above baseline overall contributions. Therefore, the presence of a social link only leads to greater cumulative productivity if the developer also has prior experience in the project's languages.

The continuous counterparts to the above independent factors are significant in this model, but only the increase in the past social l ink strength has a noticeable effect on the odds of increased productivity. As opposed to the small negative effect on productivity in the initial period (see RQ2), in this cumulative model, one unit of increase in the link strength to members of the project gives a slight benefit to productivity of 1.2% higher odds.

In summary, the answer to RQ3 is that in the long-term, for the overall contributions to a new project one's social links and prior experience matter greatly. In fact, developers with prior social links in the new project have much higher odds of having higher productivity when they also have prior experience, with an increase over baseline of between 29.5% and 54.3%. However, without prior experience, social links have a slightly detrimental effect to cumulative productivity. In aggregate, as the strength of social connection to a project grows, there is a slight growth in cumulative productivity.

Table 3: **Odds ratio coefficients from a negative binomial model for the number of total file commits to a project. [16]**

	# cumulative_file_changes
Dependent Variable:	Odds ratios
# initial_file_changes	1.059^{***}
	(0.001)
has_prior_links	0.904^{*}
	(0.043)
has_prior_exp	1.077^{**}
	(0.027)
prior_link_strength	1.012^{***}
	(0.0002)
prior_exp_strength	1.000^{**}
	(0.00000)
# prior_projects	0.999^{***}
	(0.0001)
# total_file_changes	1.000^{**}
	(0.00000)
authors_prior_to_join	1.000^{***}
	(0.00000)
is_founder	1.630^{***}
	(0.016)
time_period_joined	0.848^{***}
	(0.002)
has_prior_exp: has_prior_links	1.433^{***}
	(0.045)
Intercept	12.775^{***}
	(0.027)
Observations	39,251
Log Likelihood	−128,252.500
θ	0.832^{***} (0.006)
Null deviance:	63192
Residual deviance:	41733
Akaike Inf. Crit.	256,529.100

Note: $^{*}p < 0.1$; $^{**}p < 0.05$; $^{***}p < 0.01$

4.4 Comparing the Initial and Overall Models

The significance of the interaction term in RQ3 indicates that developers have significantly higher odds of above baseline overall contributions when they have both prior experi-

ence in the dominant languages of the project they are joining and prior social links to team members of the project. Thus, a model without the interaction term would be different substantially, which was not the case in RQ2.[4]

The seeming contrast between the effect of link strength increase being negative in the initial period and positive cumulatively is interesting. Arguably, initial periods are when a lot of the "learning the ropes" happens, and developer's joining new projects incur coordination and communication overhead related to the number of their social links in the project, as they are trying to navigate the complexity of the whole project. Cumulatively though, once the "dust settles", it is reasonable to think of one's overall contributions to a project as being more focused on smaller project modules, vs. learning the whole complex environment through their social connections. Thus, in the longer run, having more prior social connections in a project can pay dividends, as they would serve as the interface points to different project parts/modules.

The difference in the magnitudes of the *has_social_links* effects in the initial and cumulative models may suggest that social connections can pull developers to contribute for a while, but that their effect wanes in the longer term. This underlines social links as a very important recruiting factor. It is possible that this effect also carries over into the realm of retention. We noticed that when the social connection strength increases there is a slight increase in cumulative productivity. However, further investigation is needed; we leave that for future work.

5. THREATS TO VALIDITY

Data Gathering: Due to its size and the limitations on our computational and human resources, it was not feasible to mine GitHub in its entirety. Instead, we used a sample of developers who have made significant contributions to projects, as described earlier. This is, by definition, a biased set of developers. However, this is also a set of people who have participated in multiple projects, and thus can be analyzed with our methodology. They are also people who would have, arguably, made multiple social links while participating in such migrations.

We considered only commits to projects that appeared in the main repository and not those that stayed only in forks. As it is very easy to fork a project on GitHub, we believe restricting our study to commits accepted to the main branch are a better measure of meaningful contributions to the project. Ascribing meaning to these local commits is more difficult, but a model that successfully incorporates these may have different results. Likewise, we consider only the author fields of commits when measuring contributions, as we are interested in the original author of the source code, even if that author contributed the code originally in a fork.

To truly measure the preferences for joining projects, we would need to gather data on all projects on GitHub to see all projects potentially joinable projects where a developer had some connection to the team members. As we have data on only a subset of projects, our study is only an estimate.

Limitations in Methods: Our measures of identifying the language of a file is based on file extensions and language labels on repositories. More accurate measures could examine the contents of the file itself to make a more accurate measure of the file types, though we tried to be conservative in labeling a file a member of a specific language. About 1% of the files could not be identified as either a specific source language or non source code files, and were categorized as ambiguous. We excluded these from our study, but we do not believe the impact is significant.

Moreover, we excluded non source code contributions from our study. It is possible that a developer may begin contributing to a project exclusively in non source code files and then begin contributing to source code later. In this case our labeling of the initial period is the initial period of source code contribution in the project.

Resolving aliases of a single user from git commit information can only give us an estimated set of commits for a unique user id. If a developer uses multiple accounts with dissimilar names and emails, unifying all their commits may not be possible.

Our measure of social connections between developers is incidental and perhaps overly simple. We try to control for this by normalizing by the number of other authors in each project shared. A more comprehensive metric would involve communication correspondence between developers.

6. CONCLUSIONS

We sought out to elucidate the link between the socio-technical relationships among developers in the GitHub ecosystem and their migration to new projects. Our findings, that developers preferentially choose to join projects where they have prior working relationships, are certainly not unexpected. Awareness of this fact may be utilized when recruiting or retaining developers.

The effects of past linguistic experience and prior relationships on productivity, both in the initial joining period and cumulatively, are nuanced and thus intriguing. The improved productivity provided by social connections in linguistically familiar environments shows that developers not only preferentially join projects to which they are socially connected, but also contribute more (cumulatively) in the presence of stronger social ties.

However, engaging newcomers meaningfully is a key concern for both new and established projects, and our results suggest that in unfamiliar environments, the presence of past social ties alone may not be enough to lead to continued contributions in the long term. Therefore, additional measures may be needed to encourage developer retention.

Of course, given the complex social environment of GitHub and the distributed nature of `git` itself, associations between prior social contacts, language environment, and productivity may be a reflection of an underlying causal network that incorporates these and other factors. Models of this dynamic network would assist both recruiters to projects and developers considering what projects to join; our research is a step towards understanding factors relevant to such a system.

7. ACKNOWLEDGMENTS

The authors are partially supported by NSF under grants 1247280 and 1414172. We also acknowledge support from UC Davis, the members of our DECAL research group for their valuable comments, and particularly Baishakhi Ray for providing the initial git log parsing scripts and giving feedback on the paper.

[4]The interaction term is not usually interpreted in isolation, without the corresponding comprising factors.

8. REFERENCES

[1] A. Begel, J. Bosch, and M.-A. Storey. Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. *Software, IEEE*, 30(1):52–66, 2013.

[2] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.

[3] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? Immigration in open source projects. In *MSR*, pages 6–6. IEEE, 2007.

[4] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *ESEM*, pages 2–11. ACM, 2008.

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

[6] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison. Self-organization of teams for free/libre open source software development. *Information and Software Technology*, 49(6):564–575, 2007.

[7] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *CSCW*, pages 1277–1286. ACM, 2012.

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

[9] S. Faraj and L. Sproull. Coordinating expertise in software development teams. *Management Science*, 46(12):1554–1568, 2000.

[10] M. Gharehyazie, D. Posnett, B. Vasilescu, and V. Filkov. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empir. Softw. Eng.*, pages 1–36, 2014.

[11] G. Gousios and D. Spinellis. GHTorrent: Github's data from a firehose. In *MSR*, pages 12–21. IEEE, 2012.

[12] R. Guimera, B. Uzzi, J. Spiro, and L. A. N. Amaral. Team assembly mechanisms determine collaboration network structure and team performance. *Science*, 308(5722):697–702, 2005.

[13] J. Hahn, J. Y. Moon, and C. Zhang. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391, 2008.

[14] D. A. Harrison, K. H. Price, and M. P. Bell. Beyond relational demography: Time and the effects of surface-and deep-level diversity on work group cohesion. *Academy of Management Journal*, 41(1):96–107, 1998.

[15] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: A taxonomical study of large commits. In *MSR*, pages 99–108. ACM, 2008.

[16] M. Hlavac. *stargazer: LaTeX/HTML code and ASCII text for well-formatted regression and summary statistics tables*. Harvard University, Cambridge, USA, 2014. R package version 5.1.

[17] C. Jensen and W. Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *ICSE*, pages 364–374. IEEE, 2007.

[18] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining GitHub. In *MSR*, MSR 2014, pages 92–101. ACM, 2014.

[19] K. Lakhani and R. Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In *Perspectives on Free and Open Source Software*. MIT Press, Cambridge, 2005.

[20] K. R. Lakhani and E. Von Hippel. How open source software works: "free" user-to-user assistance. *Research Policy*, 32(6):923–943, 2003.

[21] J. M. Levine and R. L. Moreland. Progress in small group research. *Annual Rev. Psych.*, 41(1):585–634, 1990.

[22] A. Lima, L. Rossi, and M. Musolesi. Coding together at scale: Github as a collaborative social network. *CoRR*, abs/1407.2535, 2014.

[23] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: activity traces and personal profiles in GitHub. In *CSCW*, pages 117–128. ACM, 2013.

[24] T. Mens, M. Claes, and P. Grosjean. ECOS: Ecological studies of open source software ecosystems. In *CSMR-WCRE*, pages 403–406. IEEE, 2014.

[25] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik. Studying evolving software ecosystems based on ecological models. In *Evolving Software Systems*, pages 297–326. Springer, 2014.

[26] R. L. Moreland and L. Thompson. Transactive memory: Learning who knows what in work groups and organizations. *Small Groups: Key Readings*, pages 327–346, 2006.

[27] D. A. Owens, E. A. Mannix, and M. A. Neale. Strategic formation of groups: Issues in task performance and team member selection. In *Composition*, pages 149–165. Elsevier, 1998.

[28] J. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, 2007.

[29] I. Rivals, L. Personnaz, L. Taing, and M.-C. Potier. Enrichment or depletion of a go category within a class of genes: which test? *Bioinformatics*, 23(4):401–407, 2007.

[30] G. Robles and J. M. Gonzalez-Barahona. Contributor turnover in libre software projects. In *Open Source Systems*, pages 273–286. Springer, 2006.

[31] G. R. Salancik and J. Pfeffer. A social information processing approach to job attitudes and task design. *Administrative Science Quarterly*, pages 224–253, 1978.

[32] I. Steinmacher, T. U. Conte, M. Gerosa, and D. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *CSCW*, pages 1–13, 2015.

[33] I. Steinmacher, M. A. Gerosa, and D. Redmiles. Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective*, 2014.

[34] D. Terdiman. Forget LinkedIn: Companies turn to GitHub to find tech talent. `http://www.cnet.com/news/forget-linkedin-companies-turn-to-github-to-find-tech-talent/`, 2012.

[35] B. Uzzi and J. Spiro. Collaboration and creativity: The small world problem. *American Journal of Sociology*, 111(2):447–504, 2005.

[36] B. Vasilescu, V. Filkov, and A. Serebrenik. Perceptions of diversity on GitHub: A user survey. In *CHASE*, 2015.

[37] B. Vasilescu, D. Posnett, B. Ray, M. G. J. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov. Gender and tenure diversity in GitHub teams. In *CHI*, pages 3789–3798. ACM, 2015.

[38] B. Vasilescu, A. Serebrenik, and V. Filkov. A data set for social diversity studies of GitHub teams. In *MSR*, 2015. to appear.

[39] J. M. Ver Hoef and P. L. Boveng. Quasi-poisson vs. negative binomial regression: how should we model overdispersed count data? *Ecology*, 88(11):2766–2772, 2007.

[40] G. Von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: A case study. *Research Policy*, 32(7):1217–1241, 2003.

[41] M. Weiss and G. Moroiu. *Ecology and dynamics of open source communities*, pages 48–67. IGI Global, 2007.

[42] M. Weiss, G. Moroiu, and P. Zhao. Evolution of open source communities. In *Open Source Systems*, pages 21–32. Springer, 2006.

[43] Q. Xuan and V. Filkov. Building it together: Synchronous development in OSS. In *ICSE*, pages 222–233. ACM, 2014.

[44] Y. Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *ICSE*, pages 419–429. IEEE, 2003.

[45] M. Zhou and A. Mockus. Does the initial environment impact the future of developers? In *ICSE*, pages 271–280. ACM, 2011.

[46] M. Zhou and A. Mockus. What make long term contributors: Willingness and opportunity in OSS community. In *ICSE*, pages 518–528. IEEE, 2012.