Contents lists available at ScienceDirect

Research Policy

journal homepage: www.elsevier.com/locate/respol

What makes the right OSS contributor tick? Treatments to motivate high-skilled developers

Inna Smirnova[a,*], Markus Reitzig[b], Oliver Alexy[c]

[a] School of Information, University of Michigan, 105 S State St., Ann Arbor, MI 48109-1285, United States
[b] Strategic Management Subject Area, Department of Accounting, Innovation and Strategy, University of Vienna, Oskar-Morgenstern-Platz 1, Vienna 1090, Austria
[c] TUM School of Management, Technical University of Munich, Lichtenbergstr. 6/II, Garching b. München 85748, Germany

ARTICLE INFO

JEL codes:
O36
M52
L17

Keywords:
Open source software
Contributor skill
Motivation
Organizational design
Contributor effort

ABSTRACT

We study how OSS project owners can manage their repositories so as to motivate particularly high-skilled coders to exert continuous effort after joining a project. Drawing on literature from personnel economics, we lay out how coders' skill level affects their selection for a focal project in the first place. In turn, we theorize how project-specific norms and quality aspirations that developers learn about after joining an OSS project represent treatments that varyingly entice developers to contribute more code conditional on their skill level. Based on a custom-tailored dataset merging GitHub and Stack Overflow data for almost 50,000 contributor-project-month observations, we find that repository owners are able to motivate their most talented volunteer contributors when they (1) show no visible commercial orientation while managing their projects, (2) show generosity in accepting external contributions, and (3) provide fast feedback. We discuss implications for research and practice in the fields of community-based organizations like OSS as well as personnel economics.

1. Introduction

Organizational forms emphasizing self-selection, such as open source software (OSS) development, have become increasingly prominent in the literature on the organization of innovation (Puranam et al., 2014; Raveendran et al., 2021). Given how OSS development seemingly departs from traditional modes of governance, a broad array of studies has looked at what motivates individuals to join to such efforts (see, e.g., Bagozzi and Dholakia, 2006; Ghosh et al., 2002; Howison and Crowston, 2014; von Krogh et al., 2012). This literature finds that OSS developers infer from observable project features—such as programming language, intended goal, or license (Belenzon and Schankerman, 2015; Bonaccorsi and Rossi, 2003; Crowston and Scozzi, 2002; Fang and Neufeld, 2009; Fershtman and Gandal, 2007; Santos et al., 2013; Sen et al., 2008; Stewart et al., 2006; Stewart and Gosain, 2006; Subramaniam et al., 2009; von Krogh et al., 2003)—whether they will be able to satisfy any of the different motivational desires they may have (Benbya and Belbaly, 2010; David and Shapiro, 2008; Hars and Ou, 2002; Hertel et al., 2003; Jeppesen and Frederiksen, 2006; Lakhani and Wolf, 2005; Lee and Cole, 2003; Lerner and Tirole, 2002; Roberts et al., 2006; Shah, 2006): solving a problem they face (use-need), advancing their career or status among peers (extrinsic motivation), or deriving some form of enjoyment from working on the project per se (intrinsic motivation). In sum, if developers expect their idiosyncratic motivation can be satisfied at an effort level lower than their expected opportunity costs (i.e., the benefits of joining the specific project are higher than the expected costs, and also higher than the net benefits of joining another OSS project or doing any other activity), they should join a project.

Notwithstanding the important insights this literature has produced, two important and interconnected questions stand largely unanswered. First, we note that prior work has focused on what motivates averagely gifted OSS developers to contribute to a specific project. The question of why potentially *good* or *excellent* contributors—those with maximal skills matching the project needs—would devote more of their time to one OSS project than to another has received scant attention at best (Belenzon and Schankerman, 2015; Ghosh et al., 2002; Ho and Rai, 2017; Howison and Crowston, 2014; Roberts et al., 2006; von Krogh et al., 2012; Wasko and Faraj, 2005; Xu et al., 2009). *Keeping* highly skilled developers motivated *after* joining is OSS project managers' ultimate goal. After all, it is they who should make the most substantial contributions (Cosentino et al., 2017; Kalliamvakou et al., 2016; von Krogh et al., 2003; Zhang et al., 2013). Similarly, second, we note that works studying the effects of project features on OSS developers' motivation also focuses on project selection (Fitzgerald, 2006; Howison and

* Corresponding author.
E-mail addresses: innas@umich.edu (I. Smirnova), markus.reitzig@univie.ac.at (M. Reitzig), o.alexy@tum.de (O. Alexy).

https://doi.org/10.1016/j.respol.2021.104368
Received 27 July 2020; Received in revised form 26 July 2021; Accepted 6 September 2021
Available online 4 October 2021
0048-7333/© 2021 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Crowston, 2014; Sen et al., 2008). Yet, developers can only learn about project-specific norms and quality aspirations *after* they have joined a project (Ho and Rai, 2017; Mingers and Walsham, 2010; von Krogh et al., 2012). Designing the respective project features consciously should hence matter—to theory and OSS practitioners alike—if they would varyingly affect the expected attainable need, status, or learning outcomes and opportunity costs of developers of different skill levels. Accordingly, in this paper, we ask: what makes the *right* developer tick, and how may managers of OSS projects entice these individuals to be productive *continuously* through the design of their projects?

To ground our argument, we borrow from the longstanding debate in the field of personnel economics that addresses the interplay between worker sorting and personal skill (Cadsby et al., 2007; Dohmen and Falk, 2011; Lazear, 2000a, 2000b).[1] Seen through this lens, current OSS literature has shown that developers *select* into projects based on observable design parameters, such as a fit between their ideology and projects' license choice (Belenzon and Schankerman, 2015; Fershtman and Gandal, 2007; Sen et al., 2008; Stewart et al., 2006; Stewart and Gosain, 2006), or between their status ambitions and a project's (larger) size (Chengalur-Smith et al., 2010; Hann et al., 2013). What remains unclear, however, is how such *selection* into OSS projects affects skill-based *sorting* eventually—that is, how individuals' motivation and skill jointly lead them to pick a specific project—and, in turn, how project managers should *treat* developers who have joined a project to leverage the full potential of the best available talent.

We begin from the premise that individuals will, among other things, select into OSS projects for which they have at least some—albeit varying levels of—skills. This is because a certain level of competence is required to enjoy the programming task (and to be motivated intrinsically) (Sanders, 1998; Shah, 2006; von Hippel and von Krogh, 2003) and to produce outputs that allow for eventual status and career advances (and to be motivated extrinsically) (Crowston and Scozzi, 2002; Hann et al., 2013; Lee et al., 2003; Subramaniam et al., 2009). Motivation and skill level hence jointly determine the benefits and opportunity costs that a developer would expect from joining a project.

In turn, after joining, developers will find themselves treated by newly discovered project-level design features that embed the projects' norms and quality aspirations. Yet, this treatment effect should vary across developers of different skill levels conditional on how they see the benefits and opportunity costs from their continuous involvement with the project impacted. In particular, we argue that because high-skill individuals have a higher incentive to see their actual code contributed rather than learn from feedback (i.e., different benefits) as well as more outside options to attain their motivational benefits (i.e., different opportunity costs), they should prefer (vs. lower-skilled individuals) those projects which (a) more predictably follow the meritocratic standards of OSS, (b) more generously accept contributions, and (c) share feedback faster.

For our tests, we draw on an originally compiled and custom-tailored dataset merging information from two major software community archives—GitHub and Stack Overflow—resulting in almost 50,000 contributor-project-month observations. Our data contain information on both contributors' programming output, as measured in their proposed code patches (= pull requests) sent to other GitHub projects, and contributor skills, as measured by the expert answers they provide on Stack Overflow.

Our results provide a first insight into what makes the *right* contributor tick, and how crucial project-design parameters drive the efforts of differently skilled contributors. Conditional on the match between contributors' experience and project-specific skill requirements, we find that projects launched by founders who have a commercial orientation attract lower effort levels from more highly skilled developers, whereas a higher acceptance rate and faster feedback time increase the efforts of highly skilled developers. Building on these insights, we extend current discussions on who contributes to OSS projects (Howison and Crowston, 2014; Lakhani and Wolf, 2005; von Krogh et al., 2012). In addition, we present adaptations of formalisms from personnel economics (Booth and Frank, 1999; Lazear, 2000a, 2000b; Delfgaauw and Dur, 2007; Eriksson et al., 2009), which may inform future work studying self-selection into dispersed teams, such as the gig economy, more broadly.

2. Selection, sorting, and treatment in open source software

The field of personnel economics investigates the managerial interplay of attracting *and* continuously motivating skilled workers in firms (see, e.g., Hartog, 1986; Jovanovic, 1979; Lazear, 2001; Lazear et al., 2012). This literature illustrates that the promise of certain incentives to candidates may affect their willingness to join an organization in the first place (*selection*). How the actual incentives (*treatments*) provided to staff after joining are perceived by employees eventually depends on the prior selection, and this perception may differ across employees. A company may, for instance, be aware that offering a variable pay scheme may attract highly driven employees. What a firm seeking to hire productive folks additionally needs to know is whether one particular incentive scheme—say, piecemeal—beats another—say, a tournament—in meeting the other social preferences of productive individuals to optimally motivate them once they join (Dohmen and Falk, 2011). Only if the company knows how people *sort* across schemes, and which scheme motivates good people to exert effort after they have joined, can it advertise its compensation in the marketplace to attract top performers who will devote their time to the firm.

2.1. *Selection and sorting in OSS development*

OSS communities differ from traditional organizations in many ways (Dahlander and Frederiksen, 2012; Lakhani and Panetta, 2007; Mockus et al., 2002; Raymond, 1999; Stewart and Gosain, 2006). Contributors usually do not receive financial remuneration (O'Mahony and Ferraro, 2007), and also do not submit themselves to traditional managerial forms of task allocation; rather OSS developers self-select into projects where they volunteer their effort (Benkler, 2002; Lakhani and Panetta, 2007; Puranam et al., 2014; Raveendran et al., 2021; von Hippel and von Krogh, 2003).

Accordingly, prior research has devoted much attention to the question of what motivates a developer to join an OSS collective, and how organizations may drive these motivational stimuli to get individuals to join their projects (see, e.g., von Krogh et al., 2012, for a review)—that is, to influence *selection*. Specifically, intrinsic motivations aside (e.g., enjoyment of the creative process, learning, or a feeling of community identification), extrinsic desires such as fulfilling use-needs, gaining status within the OSS community, and visibility for career-related purposes (Jeppesen and Frederiksen, 2006; Ke and Zhang, 2010; Lakhani and Wolf, 2005; Lerner and Tirole, 2002; Roberts et al., 2006) stimulate individuals working in communities.

Given the above, so we argue, project-related skills of developers should ultimately determine a good part of coders' decisions about whether to join a specific project or not. This is because all OSS developers eventually seek to be productive, implying the target code itself or the process leading to its production are the reasons for their engagement. Specifically, individuals who hold more of the skills required to contribute

[1] The personnel economics literature distinguishes sharply between selection, sorting, and treatment. In short, individuals *select* (e.g., into work) based on observable stimuli (e.g., different financial incentive schemes). In turn, *sorting* is observed *after* selection, when designers may find that clearly discernable groups selected different variants of the stimulus, because of varying underlying characteristics or preferences (e.g., varying risk preferences, overconfidence levels). *Treatment*, in turn, implies stimuli given to individuals after their selection decision. Accordingly, for the treatment to be effective, not just the selection decision but also the sorting outcome needs to be understood by the designer. For the entire organization to be effective, designers will need to understand what happens across all three stages: selection, sorting, and treatment. While this logic has been applied primarily to standard business organizations, it is transferable to organizations of all kinds. In this paper, we present an extension for OSS projects.

to a specific OSS project should be more likely to enjoy contributing and be motivated intrinsically in turn (e.g., Sanders, 1998; Shah, 2006). Skill match should also allow individuals to produce better outputs so as to corroborate their status among peers and advance their career prospects, hence fostering extrinsic motivation (e.g., Ferraro and O'Mahony, 2012; Hann et al., 2013; Lee et al., 2003; von Krogh et al., 2003). Accordingly, we expect that higher-competence individuals should find it easier to satisfy all types of motivation (Bandura, 1977; Deci and Ryan, 1985). In the absence of other remuneration, the ability to be productive while coding thus becomes the key constraint on developers' participation,[2] and only those contributors who are somewhat qualified should therefore eventually sort across projects.

The above is not to say that all folks joining a project will be equally skilled, however. While we expect all of them to meet minimum levels of project-specific experience, we would still expect skill matching between individuals and the projects' need to vary vastly. Volunteer developers, for example, often need not meet absolute levels of output, let alone sustain a living from their participation (Benkler, 2002; Hann et al., 2013), so they can afford their skills to match onto the project less than perfectly initially, particularly when what they really aspire is to build up the requisite skills through their very project participation (Benbya and Belbaly, 2010; Lee and Cole, 2003; Lakhani and von Hippel, 2003). Coders seeking to boost their professional CV by contributing on a project may not enjoy the same degrees of freedom, however, and will seek to make meaningful contributions, requiring them to bring all the skills needed from the start.

In sum, motivation and skill may not only independently affect an individual's decision to sort into a given project. The interactions between these two effects will become more complicated when discussing the implications of how project managers eventually want to *treat* different types of developers. Notably, project managers will appreciate an understanding of the linkage between motivation and skill, so as to entice more contributions from their stronger contributors. That, in turn, requires a more nuanced understanding of the interaction between skill, motivation, and the eventual opportunity costs of coding that developers may be facing.

2.2. Untangling treatment effects in OSS: the role of project-level norms and aspirations

In general, opportunity costs should fall the more productive developers can be while coding. That means, the more individuals can attain their motivational goals on a given project (i.e., they expect a net benefit of continuing to work on the project vs. doing any other activity and vs. working on another project), the more inclined they should be to voluntarily *select* into developing OSS in general, and a specific project in particular. Following our above argument, we would thus expect that individuals would select onto those projects where they minimize their opportunity costs by leveraging their extant skill base (Lazear, 2000a, 2000b). At the same time, the actual opportunity costs may vary tremendously among the differently motivated peers. The distribution of skills among coders on a given project should thus vary greatly, the more individuals join for a variety of reasons.

The challenge for OSS project owners now becomes motivating those with the highest skill base best, as their opportunity costs will likely be highest. This is because developers of higher skill levels (1) should expect to be productive immediately (given they have all requisite skills) and (2) more likely have outside options (given that they have a skill set they can redeploy). Accordingly, project owners need to create incentive-compatible[3] *treatments* that reduce opportunity costs of code production for higher-skilled volunteers who have selected onto the project more than for lower-skilled ones.

Such treatments refer to those actions which project founders can take and which will only affect developers after joining, such as all actions related to the norms and quality aspirations of the project.[4] As von Krogh and colleagues (2012) explain, developers may truly learn about these project-specific aspirations only after joining it (see also, e.g., Ho and Rai, 2017; Lee and Cole, 2003; Mingers and Walsham, 2010), explaining perhaps why developers spend significant time "lurking" on a project before actively contributing (Nonnecke and Preece, 2003; von Krogh et al., 2003). In turn, like treatments in the corporate world, key choices about how to design and communicate these norms and aspirations should differently impact the expected benefits and opportunity costs of low-skilled versus high-skilled developers, so that they differently adjust their future effort levels on the focal project. Put differently, we expect that OSS developers experience project-specific norms and aspirations differently once they join the project.

Notably, this treatment effect is one that cannot be fully disentangled from the coders' skill-based selection, so we refer to it as a conditional treatment effect henceforth (see Appendix A3 for a formal derivation of our logic). In the following, we hypothesize such conditional treatment effects caused by design choices project founders made to embody their envisioned norms and aspirations, and which contributors should only be able to capture fully after having joined the project: founders' commercial intent, and the type and speed of feedback provided.

2.3. The conditional treatment effect of founders' commercial intent

In general, OSS projects are expected to adhere to norms of meritocracy, fairness, and reciprocity (Bergquist and Ljungberg, 2001; Chou and He, 2011; Daniel et al., 2018; Hann et al., 2013; Levine and Prietula, 2013; Maruping et al., 2019; O'Mahony and Ferraro, 2007; Raymond, 1999; Shah, 2006; Spaeth et al., 2015; Stewart and Gosain, 2006). The expectation is that developers and project managers share a form of common "hacker ethos" (Raymond, 1999), so that, in the end, the best code should simply win.

A key part of how strongly an OSS project is expected to follow these norms is embedded in the type of software license chosen (Belenzon and Schankerman, 2015; Fershtman and Gandal, 2007; Sen et al., 2008; Stewart et al., 2006; Subramaniam et al., 2009; West and O'Mahony,

[2] In the field of personnel economics, the *participation constraint* refers to a condition that must be met for a person to engage in a certain activity (e.g., a minimum salary needs to be paid for a person to sign an employment contract). In the OSS environment, the ability of developers to produce code is the necessary requirement for them to participate, as it is code production itself that brings about satisfaction, irrespective of why exactly they joined.

[3] In personnel economics, an *incentive compatible* environment refers to an arrangement that induces individuals to reveal their private information (e.g., skill level) and be truthful in their actions (e.g., behave in the employer's interest). An important implication is that higher-skilled workers, when being offered a contract compatible with their incentives, will find their compensation packages more attractive than the packages that lower-skilled workers would receive, and vice versa. In the context of this paper, incentive compatibility for highly skilled workers means that high-skilled coders feel that they are working on a project that is particularly rewarding to them, implying that it would be less attractive to lower-skilled workers.

[4] Self-evidently, tracing the selection of OSS developers into projects will never be as clean as with workers into a corporate environment. More often than not, developers do not formally sign on to an OSS project. They are considered contributors from the moment they first become active on a project. That first activity, however, is already an act of production (Raymond, 1999; von Krogh et al., 2003). Hence, the line between selection, sorting, and treatment—which is clearly delineated in personnel economics—is far more blurred in OSS production. Also note that developers engaged in OSS will likely never observe the entire spectrum of potential repositories in which they might productively engage. As a consequence, they will never make a fully informed choice when picking one project over another (Fitzgerald, 2006; Raymond, 1999), whereas workers in commercial markets may know the full choice set of employment opportunities, at least in some instances. In turn, the analysis of archival OSS data will never allow for a perfect selection control. We address these issues more formally in our Appendix A3.

2008). The project license is clearly communicated, and programmers can, for example, judge easily upfront whether a license is commercially oriented or not. Programmers may thus select into OSS projects given their preference for specific social and work norms that a (non)commercial license encapsulates, and how these correspond to their individual motivation. For example, many developers may strongly feel that software should be "free as in freedom" and avoid firm-run projects (Stallman, 1999). Others may seek to be visible to corporate actors who they hope will value the software product (Hann et al., 2013; Jeppesen and Frederiksen, 2006).

After joining, however, the license type per se may say little about the processes and structures through which OSS software is produced. In the end, it merely implies the form of protection or openness chosen for the output of the OSS project, and allows potential contributors to form initial expectations about the levels of motivational stimuli and productivity levels they can achieve on a specific project.

These expectations, however, may need to be revised when developers learn that a specific project might deviate from the expected levels of meritocracy, fairness, and reciprocity. Specifically, contributors may fear that these norms may begin to play a less dominant role if they realize that project founders are driven by initially hard-to-spot commercial or status ambitions (Shah, 2006), or when project founders unexpectedly move into salaried employment with a commercial firm benefiting from the OSS project to become the firm's "man on the inside" (Dahlander and Wallin, 2006). Upon newly identifying such ambitions, we suggest that some developers may expect that founders might choose, if only at the margin, to manage their OSS projects in a manner that deviates from the norms of OSS (Dahlander and O'Mahony, 2011; Mateos-Garcia and Steinmueller, 2008; Shah, 2006).[5] Below, we lay out why this expectation should lead to highly skilled programmers reducing their project-specific efforts more than less highly skilled ones.

Highly skilled programmers should be particularly interested in being productive. That is, given that they already bring a "stockpile of skills" (Shah, 2006, p. 1008), they expect a software development process that allows them to reap the motivational benefits from applying current skills rather than acquiring new ones at high opportunity costs (Aberdour, 2007; Mockus et al., 2002; O'Mahony and Bechky, 2008; O'Mahony and Ferraro, 2007; Shah, 2006; von Krogh et al., 2003). In turn, we expect that highly skilled programmers fueled by all types of motivation should reduce their effort when founders' commercial intent is revealed.

More specifically, intrinsically motivated, highly skilled developers should particularly value meritocracy as driver of the OSS programming process (Levy, 1984; Mockus et al., 2002; Nelson et al., 2006; Raymond, 1999). They should be taken aback by any non-coding-related reason to deviate from these norms (Mateos-Garcia and Steinmueller, 2008), in particular when that would cause their (likely higher-quality) code to be (unfairly) rejected. Similarly, extrinsically motivated, highly skilled developers may reasonably believe that founders high in commercial intent will try to catch more of the spotlight. Such developers may also fear that founders may receive (with or without founders' doing) the credit for others' work (Merton, 1968).

Less-skilled programmers, on the other hand, should more likely continue to exert effort to the OSS project even after learning of founders' newly unearthed commercial ambitions. Intrinsically motivated, less-skilled programmers, for example, should find it harder to tell whether their code suggestions were rejected because of a lack of quality or because of commercially oriented founders' different methods of selection (Siggelkow and Rivkin, 2009). Further, even when (unknowingly) being unfairly treated, they may still value to some extent that they are receiving feedback to help them achieve their goals of skill development. Extrinsically motivated, lower-skilled developers, in turn, should also be low in status in the open source community (Ferraro and O'Mahony, 2012). They would perceive any founder as higher in status than themselves and, hence, benefit from being associated with them (Barabási and Albert, 1999; Grewal et al., 2006; Mallapragada et al., 2012). Accordingly, they should be less willing to disassociate from a founder, even after their commercial intent is revealed. Rather, if the status of a developer depends completely on the founder (i.e., if their skill is very low), they may even hope to gain from the status increases for which commercially oriented founders should strive. In sum, we thus posit as follows:

HYPOTHESIS 1 (H1). The marginal effect of project-related developer skills on their project-specific programming output increases when OSS project founders do not visibly pursue commercial intents.

2.4. *The conditional treatment effect of feedback*

Beyond broader social norms guiding the OSS development process, quality aspirations and the process through which they are communicated are key to the culture of an OSS project (Ho and Rai, 2017; Mingers and Walsham, 2010; von Krogh et al., 2012). Such aspirations are strongly reflected, for example, in how a project administrator manages the code contributions that developers volunteer, in terms of both the speed of feedback provided and the eventual outcomes (i.e., whether a code contribution is accepted or not). While these dimensions may well be related,[6] they should have clearly discernable effects on contributors' continuous effort (e.g., Dahlander and O'Mahony, 2011; Dahlander and Piezunka, 2014; Ho and Rai, 2017; Moon and Sproull, 2008; Wooten and Ulrich, 2017; Zhang et al., 2013).

We note that feedback entails not only a first-order effect through which developers can learn about their own skill but also a second-order effect, meaning that developers would learn about the preferences and processes of the project. For example, Piezunka and Dahlander (2019) show how *any* feedback provided (i.e., even a rejection) increases the likelihood that individuals who submitted an idea to a firm will do so again—and do so in a way that the idea is more in line with the firm's expectations. Similarly, Riedl and Seidel (2018) elaborate how participants in the online community surrounding the T-shirt design firm Threadless do not just try to receive feedback on their own designs, but, similar to OSS developers lurking on software projects, try to learn about principles of what makes for a good submission on Threadless generally. At the same time, it is evident that volunteers can experience these first- and second-order effects, and understand and evaluate (for themselves, through an opportunity cost lens) only *after* they have contributed. Hence, like founders' commercial intent above, feedback should represent conditional treatment effect.

Specifically, we argue that if projects' quality aspirations and the processes guiding them are learned primarily through active participation, then (1) learning about how exclusive a project is contains both first-order and second-order feedback. Yet, as we lay out in the next section, not only is first-order feedback more important to low-skilled developers, they should also respond to first-order feedback differently. In turn, (2) the pace at which feedback is provided drives OSS developers' learning about the project's preferences. The more quickly developers receive *any* feedback (i.e., irrespective of whether first-order feedback is negative or positive), the faster they can learn about the

[5] A case in point is the history of Storybook (https://medium.com/protecting-storybooks-future). Note also how Linus Torvalds (and other famous hackers) were frequently the subject of conspiracy theories when in corporate employment (see, e.g., https://en.wikipedia.org/wiki/Transmeta). In turn, this would again suggest that our argument holds even if founders never actively mismanage projects—what suffices is that developers may reasonably fear that they might do so.

[6] In short, we expect that better projects should, all else being equal, be faster at providing feedback and be more selective in what code they would accept. Yet, we can control for project quality, etc., in project-level fixed effect models.

opportunity costs of contributing code in line with the project's preferences in the future. As we argue below, we expect that the importance of receiving such second-order feedback should be greater for high-skilled versus low-skilled developers.

2.4.1. The conditional treatment effect of feedback type

A project's acceptance rate is simply the share of accepted code submissions out of all submissions. Across many high-profile OSS projects (Lakhani and von Hippel, 2003; Lee and Cole, 2003), and the gig economy more broadly (Boudreau et al., 2011; Boudreau et al., 2016; Riedl and Seidel, 2018), this rate is comparatively low, with rejections far outweighing acceptances. The status gains for winning such exclusive competitions have often been identified as an important motivational driver, in particular for settings in which very highly skilled developers can openly compete against each other, such as on platforms like TopCoder or Kaggle. However, looking at OSS development more broadly, we expect a trade-off between the ability to be productive (i.e., likely having one's code accepted by a project with a high acceptance rate) and potential status gains (i.e., potentially having one's code accepted by a project with a low acceptance rate). Once the utility of developers increases more in seeing their output enter a project than it does in elevating their status by working on highly selective projects, a high acceptance rate will motivate them to contribute.

We argue that the incentive effect of having a higher expected likelihood of code acceptance will increase as a developer's skill level increases.[7] When highly skilled developers discover a project's low acceptance rate, they should increasingly realize that they may not be able to satisfy their motivational needs, and they should decrease their effort accordingly.

From an intrinsic-motivation perspective, the desire of highly skilled programmers to learn (from failure) should be much lower than their desire for positive feedback, such as through others using their software code (Shah, 2006). Accordingly, skilled developers should increasingly seek other projects in which to invest their time (Benkler, 2002; Daniel and Stewart, 2016). Similarly, even highly skilled, extrinsically motivated, status-oriented developers may react negatively to learning about lower acceptance rates. This is because programmers with an established record of performance have little need to signal the quality of their work to their peers or to third parties. Engaging with elitist community managers reputed to accept only a minority of the contributions they receive—while a nuisance to any coder—is of increasingly little value to them, since being rejected might harm their community reputation as a "coding rockstar" (Dabbish et al., 2012, p. 1283). With their status as high-skilled programmers established, they can focus on demonstrating their continued productivity and high performance (Dabbish et al., 2012) and will be able to do so ever better as the chances that a community accepts their contributions increase, all else being equal.

Less-skilled programmers, in turn, should be less discouraged by a high rejection rate. Looking at intrinsic motivation, less-skilled programmers striving for improvement will need to make costly investments in order to process the feedback they receive, anyway (Riedl and Seidel, 2018). Rather than hoping to spend time on coding exclusively (like their more highly skilled peers), they will first need to learn how to interpret and apply the input of others in order to improve. Yet, given that many aspects of quality may vary between OSS projects (von Krogh et al., 2012), investments in learning may be hard to transfer across projects. Hence, faced with such sunk costs and fewer outside options, less-skilled, intrinsically motivated developers should be more likely to continue exerting effort on a project even when rejection rates are high (Brockner, 1992; Keil et al., 2000; Staw, 1981).

Similarly, less-skilled, extrinsically motivated programmers may also stay longer on a project even if they learn of high rejection rates. This is because, for them, the potential status increases from an unlikely contribution may reasonably justify potential benefits they could anticipate through higher persistence. On one hand, this is because lower-skilled individuals will more likely overvalue their own competence (Kruger and Dunning, 1999) and hence require more time to fully grasp the impact of a high rejection rate. On the other hand, given just how much they may attain from a single, near-random successful contribution, it may even be rational for very low-skilled individuals to continue trying (Boudreau et al., 2011; Eagly and Chaiken, 1993). Eventually, even seeming to be contributing to projects with a reputation for being highly selective may help them create a (disproportional) record of their individual quality (Gousios et al., 2014; Hann et al., 2013). In combination, we thus propose the following:[8]

HYPOTHESIS 2 (H2). The marginal effect of project-related developer skills on their project-specific programming output increases as the overall acceptance rate of code on a project increases.

2.4.2. The conditional treatment effect of feedback speed

Similarly, we suggest that the timely provision of feedback may play an important role in varyingly stimulating the motivation of coders of different skill levels. Here, research on online peer production communities has illustrated contributors' susceptibility to responses they receive from community founders or administrators. Providing feedback has been shown to increase developers' motivations to contribute (Dahlander and Piezunka, 2014; Piezunka and Dahlander, 2019; Roberts et al., 2006; Zhang et al., 2013), whereas the lack of a response altogether has been interpreted to indicate poor community management (Dabbish et al., 2012; Tsay et al., 2014a), leading to a demonstrably negative effect on peer productivity (Zhu et al., 2013). Similarly, delayed responses to mailing-list postings have been associated with the detachment of potential contributors from a community (Joyce and Kraut, 2006; Kuechler, 2013).

We build on these findings to develop our third hypothesis. Mirroring our arguments leading up to H1 and H2, we suggest that there is a differential effect of feedback time on high- versus low-skilled contributors' expected benefits and opportunity costs. Receiving fast feedback should be more desirable for higher-skilled than for lower-skilled coders because the former should be able to interpret the signals the feedback entails more accurately than the latter. As we have argued above, this is because only by observing how project managers react to their code contributions can higher-skilled programmers understand precisely which quality aspirations the project follows, whether any adjustments to their efforts are necessary, and, were adjustments required, whether they would not rather spend their efforts elsewhere.[9] Lower-skilled programmers should similarly appreciate fast feedback. Yet, given that they yet lack key required skills, they will first need to invest in understanding the feedback, they will require more time processing it, and they should also interpret and implement it less aptly (Kruger and

[7] The world of publications in academic journals offers an interesting parallel. Here, H1 would imply that the editor(ial team) has a hidden agenda that would be learned only by engaging with the journal, and discovering which would disproportionally lead more highly skilled authors to seek other outlets. Our argument leading up to H2 would imply that more-established scholars should care increasingly less about publishing in elite journals—they just want to publish, or publish so that their work can be read by others.

[8] To conserve space, we have omitted the use-need argument, which would be structurally equivalent to the points made before. (1) Lower-skilled (vs. more highly skilled) programmers should both take longer to realize that their contributions will perpetually have a low acceptance chance (meaning that their use-need will not be implemented). Yet, (2) even if they were to realize this, given greater sunk costs and fewer outside options, they should persist longer.

[9] Continuing our above parallel, H3 implies that, all else being equal (i.e., authors should expect to receive the same benefits from an eventual publication, controlling for their quality level), we expect more highly skilled authors to withdraw a paper more quickly, or, at least, to not submit to this journal again, the longer the reviewing process takes.

Dunning, 1999; Riedl and Seidel, 2018). Thus, until they have actually developed the skills necessary to work on the focal project, their relative increases in productivity from fast feedback should be smaller than that of high-skilled developers. Since, as argued for H2, low-skilled developers' project-specific learning experience may be hard to transfer, and they may even become increasingly willing to wait for feedback, whereas more skilled developers confronted with rising opportunity costs would simply choose to exit the focal OSS project. In sum, we posit as follows:

HYPOTHESIS 3 (H3). The marginal effect of the project-related developer skills on their project-specific programming output decreases as the feedback time on code contributions to a project increases.

3. Data and methods

3.1. Setting

For our empirical tests, we draw on contribution records from two large software-engineering-related online platforms that allow us to measure the relationships between individual participation, individual performance, and community design. Specifically, we collected and combined data from GitHub, an OSS hosting platform, and Stack Overflow, a website featuring questions and answers (Q&As) pertaining to a wide range of topics in computer programming and software engineering.

GitHub is a large public OSS repository hosting site that provides social coding tools for software developers. Specifically, GitHub combines programming features with those typical of social networking websites; for example, GitHub permits developers to create personal website profiles and to make their profile information and platform activity feed and created code artifacts openly visible and accessible to other GitHub users and outside individuals (Dabbish et al., 2012; Marlow et al., 2013; Tsay et al., 2013). Within GitHub, developers can join existing repositories through forking them and having their code adjustments merged with the original repository.[10] To that end, users first copy the content of the original repository to their own local machine. They can then make their independent code changes and, if desired, send a so-called pull request with their proposed changes to the original repository owner (Kalliamvakou et al., 2016). Original repository owners have the authority to decide whether to include those code changes in the main branch of the repository (by merging the pull request) or to reject them. In addition to pull requests, contributors can begin discussions on existing projects by opening an issue that can relate, for example, to bug finding, a code enhancement request, or general feedback and comments.

Stack Overflow is a Q&A online community primarily used by computer programmers. Community members can ask coding-related questions or provide relevant answers and technical solutions to help their peers. In turn, community members may also give and receive upvotes or down-votes on their postings depending on whether a contribution was perceived as clear, well structured, and overall helpful, leading to vetted, individual-level and topic-specific track records of expertise on the Stack Overflow platform.

Both the GitHub and the Stack Overflow communities allow anyone to join who complies with the policy of a transparent working environment with freely shared user-submitted content with the entire community. Such a joint community value-creation ideology encourages contributors of both platforms to help solve common problems, share responsibility and knowledge, reciprocate expertise-based feedback, and contribute to the community's growth (Marlow and Dabbish, 2013).

Jointly, the two hosting sites contain all data required for testing our predictions. First, both sites permit the measurement of individuals' actions—whether answers provided on Stack Overflow to measure skill or, say, sent pull requests on GitHub to account for coding output. Second, and crucially, by merging data from the two sources, we can compute measures pertaining to the match in skills that an individual possesses and those required for a given project *without encountering endogeneity problems* that would arise when computing the measures from a single source alone. We do so by contrasting the variation in the programming language skills of an individual as attainable from their Stack Overflow account and the coding language skills needed to engage in a given GitHub project. Below, we describe how we built this joint dataset. In turn, we discuss and correct for potential sample-selection distortions resulting from our sampling procedure in the section "Alternative explanations and robustness checks" in the "Results" section of this paper.

3.2. Sample construction

We downloaded publicly accessible GitHub data (https://developer.github.com/v3/) through the GitHub API. First, we collected basic data on all repositories created on GitHub since its inception (29 October 2007) until the end of 2009, which yielded 146,676 repositories founded by 51,557 unique users. Second, we enriched these data by downloading 622,618 pull-request and issue activities performed on these repositories by 144,210 GitHub users for a period from the date of each individual repository's creation until June 2016 (end date of our data collection). Together, we obtained data on 769,294 founding, pull-request, and issue activities performed on 146,676 repositories by 179,356 unique[11] GitHub users.

We collected Stack Overflow data in two ways. First, we downloaded the archive officially released to the public by Stack Overflow on 13 June 2016 (https://archive.org/details/stackexchange). These data include the official dump of Stack Overflow archival activities from the platform's inception (31 July 2008) until 12 June 2016. Second, we downloaded the latest official dump (August 2012) of the Stack Overflow archive that still contained email hash numbers of users—important information required for the merge of GitHub and Stack Overflow data (see below)—courtesy of the International Working Conference on Mining Software Repositories in 2013 (http://2013.msrConferenceorg/challenge.php).

Using the Stack Overflow user identification number as key, we merged the two data dumps to obtain the full activity data from the hosting site's inception until 12 June 2016, including email hashes for all those users registered on the website before August 2012. This left us with data on 19,090,959 answers provided by 5,677,258 website unique users. Of these users, 1,294,658 were identifiable via their email hash information, accounting for 12,755,110 answers observed.

3.3. Matching GitHub and Stack Overflow data

To match user profiles from GitHub and Stack Overflow, we drew on users' email addresses. While leading to omissions when GitHub users chose not to voluntarily reveal their email addresses, this approach appears to be the only feasible way to merge the two sources (see also Badashian et al., 2014; Vasilescu et al., 2013).

Of our GitHub data sample, 63,663 users provided email addresses that we could then match against the Stack Overflow email hashes. We

[10] Self-selection of users into OSS projects is the dominant mode of task allocation on GitHub (see Kalliamvakou et al., 2014). GitHub also hosts private repositories, which software corporations may use as platforms to coordinate their co-workers, and authoritative task allocation may occur. Our data include public repositories only, however.

[11] The same GitHub user can both found their own software repository and contribute to it; therefore, there are duplicate user records when summing all repository-specific activities.

computed the respective MD5 hash numbers, compared those with the existing Stack Overflow email hashes, and included users with identical values in a final list. From this set, we eliminated user duplicates, such as by adding up all activities of a GitHub user who had multiple Stack Overflow profiles registered to the same email address. This left us with a sample of 21,344 unique users whose activities we can trace on both platforms (see Section 4.1. for a discussion of sample-selection bias).

To map these users' activities on the two platforms, we created one joint classification system of user activities. Each GitHub repository stipulates the programming language to be used by the developers working on it. We focused on 55 programming languages present on GitHub and used them as base categories for classifying the answers that a user provided on Stack Overflow. Each Stack Overflow answer corresponds to a particular question tagging a particular topic (e.g., <sql-server>, <mysql>, <ruby>).[12] We manually matched Stack Overflow answers using the tags of their corresponding questions to one of the 55 base programming-language categories present on GitHub, drawing on the help of an experienced programmer. Eventually, we managed to categorize 49.5% (425,442 of 859,236) of the answers that all users from our matched data sample had given until June 2016. The general Stack Overflow tags (e.g., <algorithm>, <data-types>) could not be categorized as referring specifically to any of the 55 programming languages used on GitHub; thus, we omitted their corresponding answers from our final dataset.

The final dataset we deploy for our statistical analysis consists of 163,003 GitHub founding, pull-request, and issue activities performed on 35,916 repositories by 21,344 users registered on both GitHub and Stack Overflow. The data have an unbalanced repository-user panel structure. Time variance is captured at the repository-user level, as each GitHub activity is timestamped upon creation. Each Stack Overflow answer is timestamped upon posting, allowing us to capture time variance also at the answer-user level.

3.4. Variables

3.4.1. Dependent variable

Our main dependent variable is *contributor programming output*, measured as the monthly (logged) number of pull requests (plus one) that a user sent to someone else's repository in GitHub. Pull requests contain original programming code provided by the focal user that they request be integrated into an extant GitHub repository.[13] Sending a pull request to the owner of an extant repository is considered the prime method of seeking to contribute in the GitHub environment (McDonald and Goggins, 2013; Tsay et al., 2014a; Tsay et al., 2014b). It represents a code *output* measure, in that contributors deem their work complete enough to be considered for approval by a third party (Marlow and Dabbish, 2013). To test H1 through H3, we computed the measure at the contributor-repository-month level, which left us with 68,481 observations of self-selection of 18,003 platform users into existing extant GitHub repositories.

3.4.2. Independent variables

It is a common practice in software engineering to measure the area of expertise of software developers by their experience and proficiency in different programming languages; for example, focusing on programming in "web-heavy" languages like Python and JavaScript can signal strong expertise in web development (Marlow and Dabbish, 2013). We thus compute *skill match* as the (logged) cumulative number of Stack Overflow answers (plus one) in the programming language of a GitHub repository selected for contribution that a user has given over time (prior to the focal month) (similar to Wasko and Faraj, 2005).[14] For the tests of H1 through H3, we deploy a measure computed at the contributor-repository-month level. For the 68,481 cases of user self-selection, we could compute skill-match values for 59,551 contributor-repository-month records. Some observations were dropped from the analysis because of two main constraints we faced. First, we do not obtain data for users' answer activity on Stack Overflow if they created an account there later than the focal activity month of a corresponding GitHub contribution; thus, for some GitHub activity months, a user's skill information is unavailable. Second, not all GitHub repositories from our data sample have a clearly defined programming language mainly used for code development.

The dichotomous variable *project founder's commercial orientation*, computed at the repository level, is associated with revealed commercial intent of the project founder. The variable takes the value 1 if a repository's founder is considered to have a commercial affiliation, and 0 otherwise. To compute this variable, we check GitHub profiles of repository owners as well as the repository homepages for internet links to outside websites. We code a repository as likely having a commercially oriented founder if they provide an internet link to a '.com/.co' website (major search engines, social media platforms, and hosting sites excluded) on their personal profile or on the project's homepage (similar to Dahlander and Wallin, 2006).

The variable *project acceptance rate*—computed as a cumulative measure at the repository-month level—is associated with project characteristics that demonstrate project owners' selectivity in assessing and implementing code innovations coming from externals to the project. To measure overall project acceptance rate, we calculate the (logged) percentage ratio (plus one) of the overall number of pull requests that were integrated into the main code branch by a project owner (or administrator) to all received pull requests.

The variable *feedback provision time*—computed as a cumulative measure at the repository-month level—is measured as the (logged) average number of days (plus one) that a project owner (or administrator) needed to evaluate submitted pull requests—to accept (and merge) or reject code changes proposed by external users.

Taking into account only repository-month observations where we do have a prior stock of pull requests (to calculate the main independent variables specific to H2 and H3) for our final analysis gives us data on 49,514 distinct user-repository-month observations, 28,323 distinct repository-month level observations, and 43,695 user-level monthly observations.

3.4.3. Control variables

We control for a series of potential confounding effects—at both the user and the repository levels—to reduce the risk of biasing our results with unobserved heterogeneity.

For each user, we control for experience with a given project and with the GitHub platform overall. To this end, we control for user tenure with the GitHub platform (*user GitHub tenure*), computed as the number of months since the user joined the website until a given month (the focal activity month on any repository), and tenure (in months) with a given project (*user project tenure*). We also controlled for user tenure with the Stack Overflow platform. The measure, however, was highly correlated with *user GitHub tenure* and therefore dropped from the final analysis to avoid multicollinearity problems. In addition, we include the logged number of issues submitted by users

[12] See http://stackoverflow.com/help/tagging.

[13] Normalizing pull requests by code lines leads to too many missing values because of the data constraints we faced. Given the nature of coding, however, pure code length is often not indicative of code quality (elegant coding may take fewer lines than mundane programming; see Gousios et al., 2016; Tsay et al., 2014b, for related evidence). Therefore, we believe our dependent variable is a qualified measure of the coder's programming output on GitHub.

[14] We have scaled (divided by 10) all independent variables after logging for the regression analyses. Note, however, that Tables 1 and 2 report descriptives for and correlations between the *unscaled* logarithmically transformed variables.

(*contributor issue activity*) at the user-repository-month level to account for their contributions to a focal project other than pull requests.[15] We also control for users' overall level of programming activity on GitHub by counting the (logged) monthly number of issues and pull requests submitted to other than the focal GitHub projects (*contributor issue activity on other repositories* and *contributor programming output on other repositories* respectively), as contributors face a potential trade-off in allocating their time and attention to different activities and projects on the platform (Daniel and Stewart, 2016). Additionally, to account for the potential trade-off of coders between contributing to OSS projects owned by somebody else and establishing their own project, we include *user founding experience* (computed as the logged number of repositories owned by a user) as a control variable. We add one to all of these counts prior to logging to avoid the generation of missing cases. Furthermore, we control for *contributor rejection rate* at the user-repository-month level since the experience of getting rejected might affect a coder's decision to further contribute to a given project. We compute this measure as the percentage ratio of the number of pull requests rejected by a project owner (or administrator) to the total number of pull requests that a user submitted to a focal project. We also control for users' level of overall answer activity on the Stack Overflow platform related to programming languages other than the language required to contribute to a focal project (*contributor Stack Overflow answer activity*)—computed as the (logged) cumulative number of answers (plus one) that a user has given over time (prior to the focal month). *Contributor Stack Overflow answer activity* indicates the developer's overall level of engagement with the platform. A higher level of busyness and time spent on Stack Overflow can potentially prevent contributors from submitting many pull requests on GitHub (Anderson et al., 2013). Additionally, we account for the total number of followers a user has received on GitHub up until the focal activity month (*user GitHub popularity*) as a signal of general status within the GitHub community (Cosentino et al., 2017; Tsay et al., 2014b).

For each repository, we control for *project age*, computed as the number of months beginning from the repository's date of creation until the given month (the focal activity month on this repository). We control for *project size* by counting the logged number of contributors (plus one) affiliated with a repository. We also control for the number of issues reported on a repository (*number of issues on a project*) to capture non-programming activity occurring on the respective open source project as well as project code quality in part, since many issues relate to bugs discovered by project members and other community peers.

3.5. *Estimation model*

Given the multilevel structure of our data and the distributional properties of our variables, we deploy a series of different models to account for various deviations from standard Gauss-Markov assumptions. Specifically, to account for unobserved heterogeneity and to mitigate the risk of incurring an omitted variable bias, we run a series of *ordinary least squares* (OLS) models with *user fixed effects* and cluster-robust standard errors.

The results of a Hausman test indicate that the individual-level fixed-effects models are appropriate. Given that our fixed effect is at the individual level, this implies that our model controls for all time-invariant unobserved individual characteristics such as major level of education and professional occupation, main location, race, and native tongue, as well as factors such as individuals' original reason for joining a project (e.g., hobbyist vs. professional).

4. Results

Tables 1 and 2 report descriptive statistics for and correlations between the logarithmically transformed variables that we computed. Table 1 shows the range for the logarithmic output by a user on a given repository to lie between 0 and 4.17, corresponding to (exp(0)-1 =) 0 and (exp(4.17)-1 =) 64 pull requests in absolute terms, respectively. The average monthly output by a user on a given repository is about 1 submitted pull request, indicating the unequal distribution of low and top performers in our sample. The average user-repository skill-match value is about 14 Stack Overflow answers given in a programming language of a repository selected for contribution, but with values ranging from 0 to 5,733—indicating considerable heterogeneity among users' proficiencies with particular programming languages required for a project. Interestingly, 27% of project founders in our sample have revealed their commercial orientation. Regarding pull-request acceptance rates, the average rate within the sample is 49.4%, suggesting that some of the repositories might be much more challenging for users to get their contributions merged than others. Finally, repository owners spend on average 31.5 days to evaluate incoming extant contributions, and up to 1,077 days to make a final decision. The key variables display significant variation at both the user and the repository levels, which is desired. Correlations are low to moderate between most of the explanatory variables at the user-repository level, with the exception of those variables that are systemically tied together (e.g., *project size* and *number of issues on a project*).

Table 3 reports estimations pertaining to H1, H2, and H3. To rule out that collinearity is affecting our coefficient estimates unduly, we upward-test our models by sequentially adding parameters to a baseline model of control variables (Wooldridge, 2010). Moreover, we test the robustness of results to the exclusion of potentially collinear control variables from the regressions. Models 1 through 3 include solely user- and repository-specific control variables. In Model 4 we add a user's

Table 1
Descriptive statistics.

Variable	Mean	Std. Dev.	Min	Max
User-repository-month level (N = 49,514)				
ln(Contributor programming output+1)	0.53	0.51	0	4.17
ln(Skill match+1)	1.15	1.43	0	8.65
ln(Contributor issue activity+1)	0.34	0.43	0	7
Contributor (%) rejection rate	15.53	31.28	0	100
User project tenure (in months)	5.85	12.38	0	86.9
ln(Contributor issue activity on other repositories+1)	0.12	0.35	0	4.73
ln(Contributor programming output on other repositories+1)	0.17	0.47	0	4.06
ln(Contributor Stack Overflow answer activity+1)	1.18	1.44	0	9.34
Repository level (N = 4,894)				
Project founder's commercial orientation (dummy YES/NO)	0.27	—	0	1
Repository-month level (N = 28,323)				
ln(Project acceptance rate+1)	3.43	1.39	0	4.62
ln(Feedback provision time+1)	2.65	1.49	0	6.98
ln(Project size+1)	3.69	1.52	1.1	8.91
ln(Number of issues on a project+1)	2.73	2.36	0	9.09
Project age (in months)	50.23	20.29	8.43	113.5
User-month level (N = 43,695)				
ln(User founding experience+1)	0.63	1.01	0	5.45
ln(User GitHub popularity+1)	3.22	1.55	0	9.94
User GitHub tenure (in months)	42.34	20.91	0	113.33

[15] On GitHub, developers use issues to track problems or suggest enhancement ideas while using the repository's code. GitHub developers clearly distinguish them from pull requests: "If you don't think you can contribute back a patch, open a new *issue*. If you think you can make some patch to fix the issue, fork the repo, and make a *pull request*" (see https://stackoverflow.com/questions/9563881/etiquette-of-github-contributing-pull-requests-vs-new-issue). Hence, anyone can raise issues on GitHub, and responsible/capable developers will take them forward to fix any problems that exist with the current repository's code base. We therefore use *contributor issue activity* as a control variable rather than as an alternative dependent variable. Readers may note, however, that regressions using *contributor issue activity* as a dependent variable are less significant but point in the same direction as our main results reported in Table 3.

Table 2
Correlations.

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
User-repository-month level (N=49,514)																	
1 ln(Contributor programming output+1)	1																
2 ln(Skill match+1)	0.07	1															
3 Project founder's commercial orientation (dummy YES/ NO)	-0.04	-0.02	1														
4 ln(Project acceptance rate+1)	0.03	0.07	-0.01	1													
5 ln(Feedback provision time+1)	-0.06	0.01	-0.01	0.15	1												
6 ln(Contributor issue activity+1)	-0.54	0.03	0.03	0.05	0.04	1											
7 Contributor (%) rejection rate	0.26	0.03	-0.03	-0.15	0.01	-0.19	1										
8 User project tenure (in months)	0.17	0.19	-0.05	0.14	0.04	0.08	0.14	1									
9 ln(User founding experience+1)	0.05	0.12	-0.02	-0.04	-0.11	-0.03	0.02	0.11	1								
10 ln(Contributor issue activity on other repositories+1)	-0.04	0.07	0.01	-0.02	-0.02	0.09	-0.01	0.02	0.13	1							
11 ln(Contributor programming output on other repositories+1)	0.15	0.12	-0.03	0.01	-0.03	-0.10	0.04	0.06	0.13	0.26	1						
12 ln(Contributor Stack Overflow answer activity+1)	-0.02	0.43	-0.03	0.03	0.04	0.06	-0.02	0.05	0.01	0.07	0.01	1					
13 ln(Project size+1)	-0.09	0.10	-0.03	0.24	0.32	0.18	0.08	0.22	-0.09	-0.01	-0.03	0.03	1				
14 ln(Number of issues on a project+1)	-0.31	0.07	0.02	0.16	0.15	0.45	-0.05	0.17	-0.06	0.01	-0.08	0.02	0.73	1			
15 Project age (in months)	-0.01	0.12	-0.06	0.26	0.36	0.08	-0.01	0.29	-0.14	-0.04	-0.02	0.08	0.48	0.35	1		
16 ln(User GitHub popularity+1)	0.16	0.22	-0.06	0.08	0.02	-0.05	0.07	0.28	0.42	0.16	0.20	0.14	0.08	0.01	0.12	1	
17 User GitHub tenure (in months)	0.08	0.18	-0.07	0.21	0.21	-0.01	0.01	0.36	0.31	0.03	0.05	0.11	0.26	0.13	0.67	0.41	1

project-specific skill-match variable. As expected, we find a strong positive relationship between the skill match of a contributor for a specific project and their programming output on that project (0.18% increase in output when our independent variable ln(Skill match+1)/10 increases by 1%, *p-value* = 0.000), a finding in line with Casalnuovo et al. (2015). In Model 5 we include key independent measures of different project characteristics that account for different incentives provided to coders of different skill levels to steer their contributions. In line with our expectations, *project acceptance rate* shows a positive and significant association with user-repository monthly output (*p-value* = 0.000), whereas *feedback provision time* is negatively correlated with a single contributor's monthly programming output (*p-value* = 0.000).

Per our theorizing, in Models 6, 7, and 8 we introduce the individual interactions of *skill match* and the project-design parameters meant to varyingly incentivize effort levels of higher- versus lower-skilled coders. Model 9 tests the joint significance of all three proposed interactions in a single specification, controlling for user fixed effects. For a given user, we find a negative relationship between *skill match* and monthly *programming output* when the user self-selects into GitHub projects of founders who revealed their commercial orientation (suggesting a roughly 0.12% decrease in user programming output, *p-value* = 0.033)—just as we propose in H1. Per H2, we find a significant positive effect of *skill match* on user's monthly *programming output* when users choose projects that give them an opportunity to have their code patches accepted and their individual performance widely established. The log-log specification of our regression model implies that a 1% increase in the interaction term leads to a 0.50% increase in the (logged) monthly number of pull requests that a user sends to the repository (*p-value* = 0.006). We observe a negative relationship between *skill match* and *programming output* when a user self-selects into a project with a longer wait time for feedback on whether their proposed code will be approved for integration into the core branch (0.36% decrease in user programming output, *p-value* = 0.020), giving us support for H3.

Fig. 1a and b through Fig. 3 illustrate the marginal effects predicted in Model 9 of the regression table (Table 3) at various levels of *skill match*, ranging from its minimum value till the 90th percentile of its distribution (most of our data points are located at the lower distributional levels).

Per Fig. 1a, which illustrates the marginal effect for H1, we find that the slope of *skill match* is statistically different (*p-value* < 0.1) between projects run by founders with commercial orientation and by founders who do not pursue commercial intents at the very low and very high (above the 95th percentile of the distribution) levels of *skill match*. Fig. 1b plots the difference in predicted coding output of coders working on projects with commercially versus non-commercially oriented founders at a particular level of our skill-match independent variable. The results suggest that individuals who completely lack the skills needed for working on a given project tend to stick to founders who are interested in pursuing commercial ambitions. As coders' skill level increases, however, the output difference between two types of projects becomes more negative—top coders (compared with their low-skilled peers) exert greater effort on GitHub projects of founders who do not show commercial ambitions.

With regard to the marginal effects for *project acceptance rate* (H2) and *feedback provision time* (H3) plotted in Figs. 2 and 3 respectively, we find that the simple slopes are statistically significant at various skill-match levels and point to the expected directions: as the value of our independent variable ln(Skill match+1)/10 increases, the relationship of *project acceptance rate* on coder's output increases (*p-value* = 0.000) whereas the effect of *feedback provision time* becomes more negative (*p-value* = 0.000), suggesting the stronger effects of both feedback type and speed on higher- versus lower-skilled coders' productivity. In particular, when pull-request acceptance rate on the project increases by 10% from its average value, coder's predicted *programming output* is increased by

Table 3
User-repository monthly output and repository-specific characteristics.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DV: Contributor programming output (natural log)	User FE	User FE	User FE	User FE	User FE	User FE	User FE	User FE	User FE	Tobit
H1: ln(Skill match+1) (/10) x Project founder's commercial orientation (dummy YES/NO)						**-0.11**			**-0.12**	**-0.12**
						(0.054)			**(0.054)**	**(0.028)**
						[0.036]			**[0.033]**	**[0.000]**
H2: ln(Skill match+1) (/10) x ln(Project acceptance rate+1) (/10)							**0.44**		**0.50**	**0.55**
							(0.172)		**(0.181)**	**(0.113)**
							[0.010]		**[0.006]**	**[0.000]**
H3: ln(Skill match+1) (/10) x ln(Feedback provision time+1) (/10)								**-0.29**	**-0.36**	**-0.38**
								(0.146)	**(0.156)**	**(0.100)**
								[0.046]	**[0.020]**	**[0.000]**
Project founder's commercial orientation (dummy YES/NO)					0.00	0.02	0.00	0.00	0.02	0.01
					(0.006)	(0.008)	(0.006)	(0.006)	(0.008)	(0.005)
					[0.950]	[0.050]	[0.940]	[0.957]	[0.047]	[0.066]
ln(Project acceptance rate+1) (/10)					0.18	0.18	0.14	0.18	0.13	0.15
					(0.019)	(0.019)	(0.025)	(0.019)	(0.026)	(0.019)
					[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
ln(Feedback provision time+1) (/10)					-0.16	-0.16	-0.16	-0.13	-0.12	-0.14
					(0.023)	(0.023)	(0.023)	(0.024)	(0.024)	(0.018)
					[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
ln(Skill match+1) (/10)				0.18	0.14	0.17	-0.02	0.23	0.09	0.05
				(0.044)	(0.050)	(0.056)	(0.064)	(0.072)	(0.070)	(0.049)
				[0.000]	[0.004]	[0.002]	[0.789]	[0.002]	[0.180]	[0.357]
ln(Contributor Stack Overflow answer activity+1) (/10)			-0.19	-0.10	-0.09	-0.09	-0.10	-0.09	-0.10	-0.08
			(0.052)	(0.057)	(0.071)	(0.070)	(0.070)	(0.071)	(0.070)	(0.016)
			[0.000]	[0.080]	[0.182]	[0.188]	[0.159]	[0.185]	[0.165]	[0.000]
ln(User founding experience+1) (/10)			-0.14	-0.18	—	—	—	—	—	-0.06
			(0.386)	(0.390)						(0.031)
			[0.717]	[0.650]						[0.061]
ln(Contributor issue activity on other repositories+1) (/10)			-0.04	-0.04	0.04	0.04	0.04	0.04	0.04	-0.08
			(0.084)	(0.083)	(0.090)	(0.090)	(0.090)	(0.090)	(0.090)	(0.055)
			[0.606]	[0.655]	[0.651]	[0.623]	[0.641]	[0.655]	[0.618]	[0.160]
ln(Contributor programming output on other repositories+1) (/10)			-0.18	-0.18	-0.27	-0.27	-0.27	-0.27	-0.27	0.13
			(0.184)	(0.179)	(0.181)	(0.180)	(0.180)	(0.181)	(0.179)	(0.044)
			[0.332]	[0.307]	[0.143]	[0.142]	[0.140]	[0.143]	[0.139]	[0.004]
Contributor (%) rejection rate (/10)		0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
		(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)
		[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
ln(Contributor issue activity+1) (/10)		-4.03	-4.04	-4.03	-4.39	-4.39	-4.39	-4.39	-4.39	-5.38
		(0.209)	(0.236)	(0.236)	(0.297)	(0.296)	(0.296)	(0.296)	(0.296)	(0.048)
		[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
User project tenure (in months) (/10)		0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.03	0.04
		(0.005)	(0.006)	(0.005)	(0.006)	(0.006)	(0.006)	(0.006)	(0.006)	(0.002)
		[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
ln(Project size+1) (/10)	0.57	0.20	0.19	0.19	0.20	0.20	0.20	0.19	0.19	0.14
	(0.043)	(0.041)	(0.045)	(0.045)	(0.046)	(0.046)	(0.046)	(0.046)	(0.046)	(0.017)
	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
ln(Number of issues on a project+1) (/10)	-0.89	-0.40	-0.40	-0.40	-0.30	-0.30	-0.30	-0.30	-0.30	-0.28
	(0.034)	(0.033)	(0.037)	(0.036)	(0.038)	(0.038)	(0.038)	(0.038)	(0.038)	(0.011)
	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
Project age (in months) (/10)	0.03	0.01	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01
	(0.008)	(0.007)	(0.009)	(0.009)	(0.012)	(0.012)	(0.012)	(0.012)	(0.012)	(0.007)
	[0.000]	[0.048]	[0.012]	[0.014]	[0.436]	[0.440]	[0.421]	[0.458]	[0.449]	[0.150]
Project age (in months) (/10), squared	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)
	[0.637]	[0.697]	[0.303]	[0.305]	[0.256]	[0.260]	[0.248]	[0.272]	[0.271]	[0.114]
ln(User GitHub popularity+1) (/10)	-0.08	-0.14	-0.16	-0.20	-0.24	-0.24	-0.23	-0.24	-0.23	0.20
	(0.136)	(0.115)	(0.139)	(0.139)	(0.139)	(0.139)	(0.139)	(0.139)	(0.139)	(0.017)
	[0.575]	[0.223]	[0.259]	[0.143]	[0.086]	[0.085]	[0.092]	[0.091]	[0.098]	[0.000]
User GitHub tenure (in months) (/10)	0.04	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.00
	(0.008)	(0.007)	(0.008)	(0.008)	(0.009)	(0.009)	(0.009)	(0.009)	(0.009)	(0.002)
	[0.000]	[0.000]	[0.001]	[0.004]	[0.066]	[0.064]	[0.067]	[0.069]	[0.068]	[0.811]
Constant	0.04	0.38	0.46	0.47	0.63	0.62	0.64	0.62	0.63	0.57
	(0.048)	(0.044)	(0.056)	(0.056)	(0.058)	(0.058)	(0.057)	(0.058)	(0.058)	(0.026)
	[0.354]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]	[0.000]
Time dummies (year)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Observations	68,481	68,481	59,551	59,551	49,514	49,514	49,514	49,514	49,514	49,514
Adj. R-squared	0.17	0.31	0.29	0.29	0.22	0.22	0.22	0.22	0.22	—

Notes. Panel regressions where robust standard errors clustered at the user level are reported in parentheses and *p-values* are reported in square brackets. Note that because of the rescaling of independent variables (/10), the coefficient estimates for testing interaction effects correspond to the true coefficients multiplied by 100. Note that for multicollinearity reasons, the variable *User founding experience* gets dropped from Models 5 through 9. DV, dependent variable.

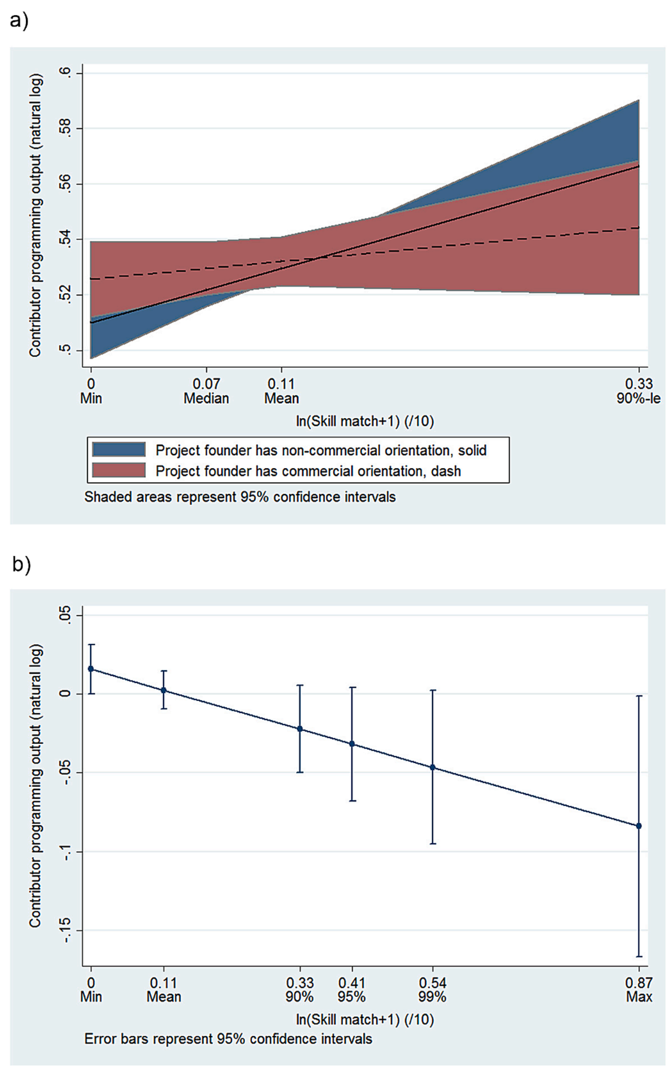

Fig. 1. (a). Predicted marginal effect of project founder's commercial orientation on user-repository monthly output (b). Difference in the predicted output of coders working on projects with commercially vs. non-commercially oriented founders.

1.2% for the averagely talented (i.e., skill-match is fixed at the mean) programmers.[16] For excellent (i.e., skill-match at the 90th percentile) developers, in turn, coder's predicted output increases by about 1.8% when *project acceptance rate* increases by 10% from its mean. In a similar vein, when *feedback* is provided 10% *faster* than the average provision time, predicted *programming output* for the averagely skilled developers increases by 0.8%, while it increases by 1.1% for very highly skilled developers.

Finally, because our dependent variable is censored toward its lower bound, in Model 10 of Table 3 we also present a *tobit* model. In addition, in results available upon request, we reran all models from Table 3 using *bootstrapped* standard errors (instead of clustering at the user level) and

[16] For the averagely skilled developers, we first predicted the output when fixing our project acceptance rate variable at its average value (while keeping all other independent variables fixed at their means), which resulted in the predicted logged programming output being equal to 0.5282. Next, we predicted the coding output given our project acceptance rate variable fixed at its (mean+10%) value, which corresponded to 0.5346. We then used these two output results to arrive at the increase in the predicted output of about ((0.5346-0.5282)/0.5282 =) 1.2%. We followed similar logic when computing all the marginal effects reported in the paper.

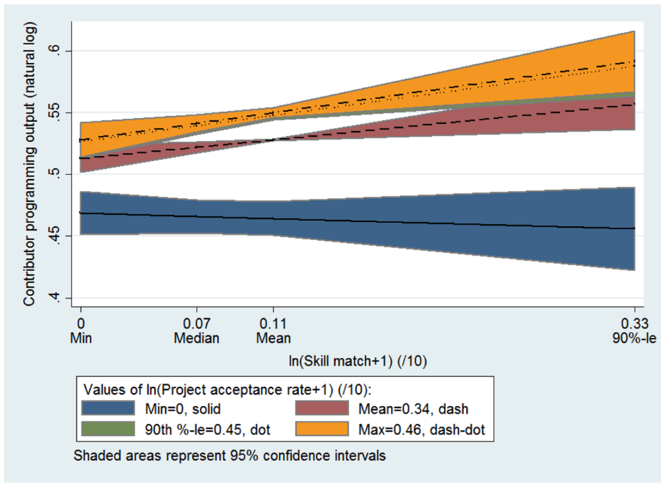

Fig. 2. Predicted marginal effect of project acceptance rate on user-repository monthly output.

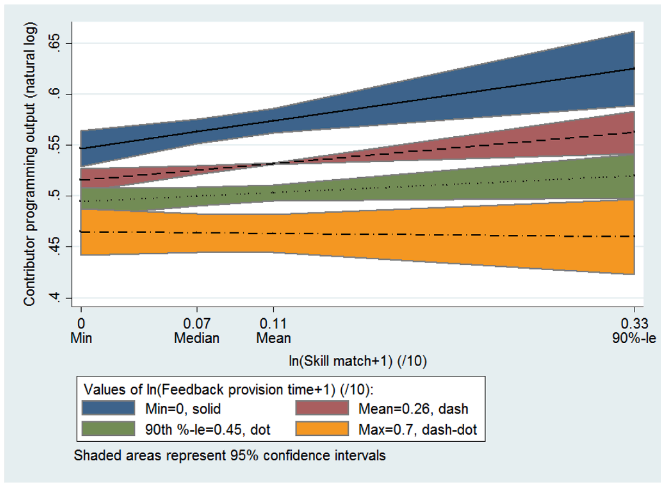

Fig. 3. Predicted marginal effect of feedback provision time on user-repository monthly output.

deploying *project-level* (instead of user-level) *fixed effects* (which control for time-invariant project characteristics such as software's intended goal or quality). Our results remain unchanged and similar in size to the findings reported herein.

4.1. Alternative explanations and robustness checks

Despite giving robust evidence supporting our main theoretical predictions, we cannot rule out that alternative explanations may account for our findings. Below, we critically discuss issues pertaining to sample construction and unobserved heterogeneity.

As described in the "Data and methods" section, our rich sample may still suffer from selection bias due to constraints when matching GitHub and Stack Overflow users to compute our independent variable *skill match*. Overall, we could match about 12% (21,344 of 179,356) of GitHub users with their Stack Overflow accounts, meaning that our results might be specific to users active on both platforms in parallel and not generalizable to the overall GitHub community. To assess this possibility, we ran two sets of further models (both reported in the Online Appendix).

In the first set, we compared the distributions of our key variables deployed in regressions reported in Table 3 within our sample and across the universe of GitHub—to the extent possible. To that end, we ran a *logistic model* for the inclusion of an observation in our sample (dummy set to 1 if a user is included in our final dataset, and 0 otherwise) at the user-repository-month level (see Online Appendix Table A1). The results show that users in our dataset have a longer tenure with GitHub, as we collected the data on users and repositories registered at the very beginning of the platform's existence; as expected, users from our sample also have more followers. Similarly, we find that our merged dataset consists of repositories that are larger and better established in terms of popularity among community members. At the same time, repository owners in our sample accept on average more incoming pull requests and assess incoming contributions more slowly than the whole population of GitHub data. Based on this model alone, it is difficult for us to determine whether the effects we obtain on our sample would be reduced or exacerbated if we were to extend our analysis to the full GitHub universe. Under the assumption of a relatively similar user-activity distribution in the full sample, we believe our findings would remain stable.

In the second set of models, we sought to assess the bias in our reported findings by explicitly modeling the sample-selection distortion we incur from being unable to match the entirety of GitHub users to Stack Overflow data. We followed a standard *two-step Heckman approach*: after estimating the selection equation at the first stage, we ran a second-stage user fixed-effects OLS model (to tease out time-invariant heterogeneity) with the inverse Mills ratio included as a regressor. The findings appear to corroborate those shown in Table 3 (see Online Appendix Table A2). Constructing samples as above, we find that interactions between *skill match* and our key variables for testing H1 through H3 in Stage 2 regressions of the Heckman model that mirrors Table 3 are statistically significant and economically comparable to the findings reported in the paper. Overall, the above findings suggest that the sample-selection distortions do not bias our results in ways that would prevent us from generalizing about our findings.

Finally, to look at issues of reverse causality, we estimated regressions using (a) lagged dependent variable(s). We find close to no indication that skill matching (the independent variable) could also be seen as driven by contributor's performance (the dependent variable).

5. Discussion and conclusion

5.1. Summary of results

In this paper, we set out to extend an established debate asking why people would join and contribute to OSS projects. Going beyond established arguments on developers' motivation to make an *initial* contribution, we set out to inquire what OSS managers can do to leverage *continuously* the best of the talent volunteering to work on their projects, irrespective of their developers' particular motivations. We theorized and found that higher-skilled as compared with lower-skilled programmers would show higher relative effort levels when (a) project founders had no visible corporate affiliation, and when (b) the project had a high acceptance rate and (c) fast feedback times. Based on these insights, in the following, we derive theoretical contributions on the organization of OSS projects, and gig-like organizations more broadly.

5.2. Implications for theory

Our first set of contributions speaks to the literature on OSS projects and their design and management. Since the original musings on what makes developers in OSS tick (e.g., Raymond, 1999), a series of academic authors have inquired into why individuals would work in OSS (e.g., Ghosh et al., 2002; Lakhani and Wolf, 2005) or on a specific OSS project (Howison and Crowston, 2014; von Krogh et al., 2012). While this work has contributed greatly to our understanding of the initial motivation of average OSS developers, it falls short of inquiring how this motivation might (a) change after joining and (b) vary between developers of different skill levels.

To answer these questions—so as to be able to say what makes the *right* developer tick—we try to shift the focus of this debate—in the words of the personnel economics literature we bring to this context—away from the initial selection toward the treatment that happens afterwards. Through this lens, we see clear indications of skill-based sorting, with individuals becoming more productive on projects to which they bring the respective skill. In line with Shah's (2006) anecdotal evidence, our results clearly suggest that experienced developers will increasingly seek OSS projects in which they may be productive, quickly.

This observation, in turn, points to the importance of looking at factors which would allow highly skilled individuals to remain productive after joining—treatments. Here, first, our results extend prior work showing that average OSS developers avoid projects that follow a commercial ideology (Elliott and Scacchi, 2008; Stallman, 1999; Stewart and Gosain, 2006). We find that when a commercial orientation is revealed post-joining, highly skilled developers show less effort on such projects. Interestingly, second, we see that this effect could not be compensated by some form of exclusivity, which could be communicated to a restrictive selection, such as in a corporate tournament. Even while high-skilled developers may *knowingly* be willing to select themselves into highly competitive environments (as in Boudreau et al., 2011; Boudreau et al., 2016; Chen et al., 2020), when they find that the chance of their efforts being accepted is decreasing, they are more likely to reduce effort. Finally, we found a strong moderating effect of feedback time on the effort of highly skilled developers. Indeed, feedback provision and speed are increasingly seen by innovation scholars as key ingredients in working successfully in open models of innovation (Piezunka and Dahlander, 2019; Riedl and Seidel, 2018). Yet, as we have argued, it is likely not the actual feedback (i.e., learning about their skills and how to improve them) that matters to highly skilled developers but learning about key project preferences.

Jointly, these three findings suggest that highly skilled developers evaluate transparency and consistency more favorably: given that they have more outside options than low-skilled developers, in discovering that the norms of a project are different from what may reasonably be expected, they are much more likely to leave. Conceptually, this would imply that in OSS-like settings that emphasize self-selection, not only should managers be transparent in communicating their project-specific norms and quality aspirations; they should try to be as comprehensive as possible in laying out these ambitions and the design features they have chosen accordingly, so that few unexpected treatments would occur post-joining. By sharing comprehensively how the project works (and, possibly, adjusting the respective process accordingly in advance), project managers may not only shift selection and sorting to their benefit; they should also maximize the odds of *both* attracting *and* retaining the most qualified individuals. In turn, such pre-emptive transparency could also help address the issue that time to provide feedback on specific contributions is a scare resource, in particular as crowds of contributors may grow infinitely larger (Piezunka and Dahlander, 2015).[17]

Our second contribution is to the field of personnel economics (Cadsby et al., 2007; Dohmen and Falk, 2011; Lazear, 2000a, 2000b). More specifically, we extend both extant theory and empirical testing approaches to a novel context that becomes increasingly relevant to the field: new work (Ollo-Lopez et al., 2010). Broadly speaking, the term captures the entirety of modern and flexible organization for work, encompassing novel approaches to the division of labor and integration of effort (Puranam et al., 2014). While the field has recently made some advances in understanding which factors beyond traditional payment schemes impact on the selection of workers into and productivity within

[17] For example, like academic journals, OSS projects could communicate their "key expectations" to volunteers publicly—including information about what will happen when code submissions do not follow such rules initially.

environments—such as technology and age (see, e.g., Schøne, 2009)—there is still a dearth of comprehensive knowledge as to how non-monetary incentive schemes influence both selection into and performance after joining specific types of organizations. Our findings, gained from a setting that shares many similarities with new models of for-profit work—such as emphasis on self-selection, the need to establish creative and maintain dispersed teams described in the literatures on information systems (Faraj et al., 2011; Wasko and Faraj, 2000, 2005) and innovation management (Bernstein et al., 2016; Criscuolo et al., 2014)—fill part of this lacuna in the theory of personnel economics. Consequently, we suggest to consider communicating non-monetary rewards such as feedback cultures and the acceptance of contributions from individuals as part of the hiring process to exploit the potential of these levers not only to increase worker productivity but also to affect their selection positively.

Moreover, we believe that our study bears some interesting takeaways for scholars working empirically on issues of hiring and treating employees. In many modern settings—e.g., crowdsourcing (Afuah and Tucci, 2012; Majchrzak and Malhotra, 2020)—the formal act of selecting into a work environment is substituted for by the de-facto act of contributing. As such, the line between selection and treatment becomes near-indistinguishable and extant methods of identification the two types of effects no longer map onto empirical realities. To these scholars, we have presented a different type of empirical approach—suggesting, in essence to interpret the interaction term of a selection and a treatment parameter as a so-called conditional treatment effect. We believe that this technique may be seen as a first step for our colleagues to better empirically analyze a new category of workplace environments in which contributors participate outside classic Simonian labor contracts (Simon, 1951). For this literature specifically, we also present some initial formalism in Appendix A3, which we hope this scholarship may extend in the future.

5.3. Limitations

Beyond potential shortcomings in our sampling and matching procedure, a few concerns remain that may limit the generalizability of our findings. Foremost, we assumed that people would select into projects based on skill, but we admit that eventual sorting may also be affected by other, unobserved variables. However, these variables should not be systematically correlated with project-related skill and as such merely add to rather than revoke our insights. Second, we are limited in our insights not just by the data we can match across our datasets but also by the period of time for which we can do so reliably. We cannot rule out that the growth of both GitHub and Stack Overflow may have coincided with, or led to, changes in coding standards or evaluation norms that could have been picked up by our key moderating variables. At the same time, given that GitHub was acquired by Microsoft in 2018, some external events happening over a longer period may have been extremely hard to account for reliably in our data.

We further concede that our commercial intent variable may fail to capture some of the nuance for which a detailed manual coding of every project page and founder profile would allow. While manual inspection shows that most of the links we have coded as symbolizing a commercial intent indeed do so, of course, we cannot fully capture the nuance of just how strong said intent would be. From a random sample of links we inspected manually, we estimate that our rate of false positives should not be higher than 10-15%. In turn, given we focus only on '.com/.co' in the domain name, we may also be missing a series of developers with clear commercial intent who link to different kinds of webpages, such as '.net' or '.de.'

Regarding our musings on feedback type and code acceptance rates, we note that our argument may not fully extend to contexts of intense status-competition, in which highly-skilled developers may actually try to differentiate from their equally skilled peers by solving particularly tough challenges, as exemplified by platforms like TopCoder or Kaggle. Similarly, we need to point out that while our conceptual argument suggests that lower rejection rates may attract better developers, empirically, there should likely be a lower bound to what kind of contributions should be accepted—an 'anything goes' attitude will likely also not attract talent.[18]

Finally, while the user-project-month structure of our panel data allows us to minimize concerns of unobserved heterogeneity, we may be falling short of fully incorporating more social explanations for coding activity, such as personal ties between project members and/or between project members and the founder (Casalnuovo et al., 2015; Grewal et al., 2006; Hahn et al., 2008).

5.4. Implications for practice

These limitations notwithstanding, we believe our results have important implications for practice. First, they should be seen as encouraging OSS project founders and repository managers to communicate clearly their project skill needs. To the extent that they ever wanted to affect selection directly, they would have to think about communicating management practices to current and future contributors. For example, commercial intent need not have a bad effect per se, as long as the commercial behavior is consistent with its communication and volunteers' perception of it. That may include firms both credibly ceding control over OSS projects they initiated (as IBM did when they transitioned to a more open Eclipse Foundation model) as well as clearly demarking spheres and modes of influence of projects in which they want to play a significant role (such as IBM's work in Linux or node.js). Second, the kind of parameters we have identified may be adjusted over time. Accordingly, should project founders and administrators identify that they do not yet attract the kind of contributors they want, they may choose to adapt key project parameters, their communication, or both. For example, our results suggest that lowering the rejection rate for contributions may help attract effort from more highly skilled people. While this sounds like a particularly interesting way to get a project off the ground, project managers would need to make sure that these individuals do not become outnumbered by less-skilled people taking the same opportunity. Finally, our insights related to feedback time further buttress the importance of communication. While evaluation of code's quality is important, project managers need to ensure that developers are not left in the dark about what happens with their contributions and why, so that developers do not take their efforts elsewhere. As projects mature and grow, that should increasingly imply that project founders need to acquire talent to support them in keeping feedback times short. Overall, we see our results as an encouragement to OSS practitioners—whether individual hobbyists or corporate teams—to become more conscientious about the organizational structures they design, and how these will impact the continuous effort of their potential contributors.

CRediT authorship contribution statement

Inna Smirnova: Conceptualization, Methodology, Investigation, Formal analysis, Visualization, Writing – original draft, Writing – review & editing. **Markus Reitzig:** Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing. **Oliver Alexy:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence

[18] We are indebted to our careful reviewers for pointing out this important set of limitations.

the work reported in this paper.

Acknowledgement

All authors would like to acknowledge funding from the Austrian Science Fund (FWF) [Grant P 25768-G16]. Special thanks go to Menaka Sattmann for her support in creating the database.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.respol.2021.104368.

References

Aberdour, M., 2007. Achieving quality in open-source software. IEEE Softw. 24 (1), 58–64.
Afuah, A., Tucci, C.L., 2012. Crowdsourcing as a solution to distant search. Acad. Manage. Rev. 37 (3), 355–375.
Anderson, A., Huttenlocher, D., Kleinberg, J., Leskovec, J., 2013. Steering user behavior with badges. In: Proceedings 22nd International Conference on World Wide Web. (Rio de Janeiro, Brazil), pp. 95–106.
Badashian, A.S., Esteki, A., Gholipour, A., Hindle, A., Stroulia, E., 2014. Involvement, contribution and influence in github and stack overflow. In: Proceedings Annual International Conference on Computer Science and Software Engineering. (Markham, Ontario, Canada), pp. 19–33.
Bagozzi, R.P., Dholakia, U.M., 2006. Open source software user communities: a study of participation in Linux user groups. Manage. Sci. 52 (7), 1099–1115.
Bandura, A., 1977. Self-efficacy: toward a unifying theory of behavioural change. Psychol. Rev. 84 (2), 191–215.
Barabási, A.L., Albert, R., 1999. Emergence of scaling in random networks. Science 286 (5439), 509–512.
Belenzon, S., Schankerman, M., 2015. Motivation and sorting of human capital in open innovation. Strat. Manage. J. 36 (6), 795–820.
Benbya, H., Belbaly, N., 2010. Understanding developers' motives in open source projects: a multi-theoretical framework. Commun. AIS 27 (30), 589–610.
Benkler, Y., 2002. Coase's penguin, or, Linux and "the nature of the firm". Yale Law J. 112 (3), 369–446.
Bergquist, M., Ljungberg, J., 2001. The power of gifts: organizing social relationships in open source communities. Inf. Syst. J. 11 (4), 305–320.
Bernstein, E., Bunch, J., Canner, N., Lee, M., 2016. Beyond the holacracy hype. Harv. Bus. Rev. 94 (7), 38–49.
Bonaccorsi, A., Rossi, C., 2003. Why open source software can succeed. Res. Policy 32 (7), 1243–1258.
Boudreau, K.J., Lacetera, N., Lakhani, K.R., 2011. Incentives and problem uncertainty in innovation contests: an empirical analysis. Manage. Sci. 57 (5), 843–863.
Boudreau, K.J., Lakhani, K.R., Menietti, M., 2016. Performance responses to competition across skill levels in rank-order tournaments: field evidence and implications for tournament design. RAND J. Econ. 47, 140–165.
Brockner, J., 1992. The escalation of commitment to a failing course of action: toward theoretical progress. Acad. Manage. Rev. 17 (1), 39–61.
Booth, A.L., Frank, J., 1999. Earnings, productivity, and performance-related pay. J. Labor Econ. 17 (3), 447–463.
Cadsby, C.B., Song, F., Tapon, F., 2007. Sorting and incentive effects of pay for performance: an experimental investigation. Acad. Manag. J. 50 (2), 387–405.
Casalnuovo, C., Vasilescu, B., Devanbu, P., Filkov, V., 2015. Developer onboarding in GitHub: the role of prior social links and language experience. In: Proceedings 10th Joint Meeting on Foundations of Software Engineering. (Bergamo, Italy), pp. 817–828.
Chen, L., Xu, P., Liu, D., 2020. Effect of crowd voting on participation in crowdsourcing contests. J. Manage. Inf. Syst. 37 (2), 510–535.
Chengalur-Smith, I., Sidorova, A., Daniel, S., 2010. Sustainability of free/libre open source projects: a longitudinal study. J. Assoc. Inf. Syst. 11 (11), 657–683.
Chou, S.W., He, M.Y., 2011. Understanding OSS development in communities: the perspectives of ideology and knowledge sharing. Behav. Inf. Technol. 30 (3), 325–337.
Cosentino, V., Izquierdo, J.L.C., Cabot, J., 2017. A systematic mapping study of software development with GitHub. IEEE Access 5, 7173–7192.
Criscuolo, P., Salter, A., Ter Wal AL, 2014. Going underground: bootlegging and individual innovative performance. Organ. Sci. 25 (5), 1287–1305.
Crowston, K., Scozzi, B., 2002. Open source software projects as virtual organisations: competency rallying for software development. IEE Proc.-Softw. 149 (1), 3–17.
Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J., 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In: Proceedings ACM 2012 Conference on Computer Supported Cooperative Work. (ACM, Seattle, Washington, USA), pp. 1277–1286.
Dahlander, L., Frederiksen, L., 2012. The core and cosmopolitans: a relational view of innovation in user communities. Organ. Sci. 23 (4), 988–1007.
Dahlander, L., O'Mahony, S., 2011. Progressing to the center: coordinating project work. Organ. Sci. 22 (4), 961–979.
Dahlander, L., Piezunka, H., 2014. Open to suggestions: how organizations elicit suggestions through proactive and reactive attention. Res. Policy 43 (5), 812–827.
Dahlander, L., Wallin, M.W., 2006. A man on the inside: unlocking communities as complementary assets. Res. Policy 35 (8), 1243–1259.
Daniel, S.L., Maruping, L.M., Cataldo, M., Herbsleb, J., 2018. The impact of ideology misfit on open source software communities and companies. MIS Q. 42 (4), 1069–1096.
Daniel, S., Stewart, K., 2016. Open source project success: resource access, flow, and integration. J. Strat. Inf. Syst. 25 (3), 159–176.
David, P.A., Shapiro, J.S., 2008. Community-based production of open-source software: what do we know about the developers who participate? Inf. Econ. Policy 20 (4), 364–398.
Deci, E.L., Ryan, R.M., 1985. Intrinsic Motivation and Self-Determination in Human Behavior. Plenum Press, New York.
Delfgaauw, J., Dur, R., 2007. Signaling and screening of workers' motivation. J. Econ. Behav. Org. 62 (4), 605–624.
Dohmen, T., Falk, A., 2011. Performance pay and multidimensional sorting: productivity, preferences, and gender. Am. Econ. Rev. 101 (2), 556–590.
Eagly, A.H., Chaiken, S., 1993. The Psychology of Attitudes. Harcourt Brace Jovanovich College Publishers.
Elliott, M.S., Scacchi, W., 2008. Mobilization of software developers: the free software movement. Inf. Technol. People 21 (1), 4–33.
Eriksson, T., Teyssier, S., Villeval, M., 2009. Self-selection and the efficiency of tournaments. Econ. Inquiry 47 (3), 530–548.
Fang, Y., Neufeld, D., 2009. Understanding sustained participation in open source software projects. J. Manage. Inf. Syst. 25 (4), 9–50.
Faraj, S., Jarvenpaa, S.L., Majchrzak, A., 2011. Knowledge collaboration in online communities. Organ. Sci. 22 (5), 1224–1239.
Ferraro, F., O'Mahony, S., 2012. Managing the boundaries of an "open" project. In: Padgett, J.F., Powell, W.W. (Eds.), The Emergence of Organizations and Markets. Princeton University Press, Princeton, NJ, pp. 540–565.
Fershtman, C., Gandal, N., 2007. Open source software: motivation and restrictive licensing. Int. Econ. Econ. Policy 4 (2), 209–225.
Fitzgerald, B., 2006. The transformation of open source software. MIS Q. 30 (3), 587–598.
Ghosh, R.A., Glott, R., Krieger, B., Robles, G., 2002. Free/Libre and Open Source Software: Survey and Study. International Institute of Infonomics, University of Maastricht, Maastricht, The Netherlands.
Gousios, G., Pinzger, M., Deursen, A.V., 2014. An exploratory study of the pull-based software development model. In: Proceedings 36th International Conference on Software Engineering. (Hyderabad, India), pp. 345–355.
Gousios, G., Storey, M.A., Bacchelli, A., 2016. Work practices and challenges in pull-based development: the contributor's perspective. In: Proceedings 38th International Conference on Software Engineering. (Austin, Texas, USA), pp. 285–296.
Grewal, R., Lilien, G.L., Mallapragada, G., 2006. Location, location, location: how network embeddedness affects project success in open source systems. Manage. Sci. 52 (7), 1043–1056.
Hahn, J., Moon, J.Y., Zhang, C., 2008. Emergence of new project teams from open source software developer networks: impact of prior collaboration ties. Inf. Syst. Res. 19 (3), 369–391.
Hann, I.H., Roberts, J.A., Slaughter, S.A., 2013. All are not equal: an examination of the economic returns to different forms of participation in open source software communities. Inf. Syst. Res. 24 (3), 520–538.
Hars, A., Ou, S., 2002. Working for free? Motivations for participating in open-source projects. Int. J. Electron. Comm. 6 (3), 25–39.
Hartog, J., 1986. Allocation and the earnings function. Empir. Econ. 11 (2), 97–110.
Hertel, G., Niedner, S., Herrmann, S., 2003. Motivation of software developers in open source projects: an internet-based survey of contributors to the Linux kernel. Res. Policy 32 (7), 1159–1177.
Ho, S.Y., Rai, A., 2017. Continued voluntary participation intention in firm-participating open source software projects. Inf. Syst. Res. 28 (3), 603–625.
Howison, J., Crowston, K., 2014. Collaboration through open superposition: a theory of the open source way. MIS Q. 38 (1), 29–50.
Jeppesen, L.B., Frederiksen, L., 2006. Why do users contribute to firm-hosted user communities? The case of computer-controlled music instruments. Organ. Sci. 17 (1), 45–63.
Jovanovic, B., 1979. Job matching and the theory of turnover. J. Polit. Econ. 87 (5), 972–990.
Joyce, E., Kraut, R.E., 2006. Predicting continued participation in newsgroups. J. Comput.-Med. Commun. 11 (3), 723–747.
Kalliamvakou, E., Damian, D., Singer, L., German, D.M., 2014. The Code-Centric Collaboration Perspective: Evidence from GitHub. Technical Report DCS-352-IR, University of Victoria.
Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D., 2016. An in-depth study of the promises and perils of mining GitHub. Empir. Softw. Eng. 21 (5), 2035–2071.
Ke, W., Zhang, P., 2010. The effects of extrinsic motivations and satisfaction in open source software development. J. Assoc. Inf. Syst. 11 (12), 784–808.
Keil, M., Tan, B.C., Wei, K.K., Saarinen, T., Tuunainen, V., Wassenaar, A., 2000. A cross-cultural study on escalation of commitment behavior in software projects. MIS Q. 24 (2), 299–325.
Kruger, J., Dunning, D., 1999. Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. J. Pers. Soc. Psychol. 77 (6), 1121–1134.
Kuechler, J.V., 2013. The Emergent Qualities of Diversity in Free and Open Source Software Communities: a Critical Review and Theoretical Discussion. PhD dissertation. Oregon State University.

Lakhani, K.R., Panetta, J.A., 2007. The principles of distributed innovation. Innovations 2 (3), 97–112.
Lakhani, K.R., von Hippel, E., 2003. How open source software works: "free" user-to-user assistance. Res. Policy 32 (6), 923–943.
Lakhani, K.R., Wolf, R., 2005. Why hackers do what they do: understanding motivation and effort in free/open source software projects. In: Feller, J.R., et al. (Eds.), Perspectives on Free and Open Source Software. MIT Press, Cambridge, MA, pp. 3–21.
Lazear, E.P., 2000a. The power of incentives. Am. Econ. Rev. 90 (2), 410–414.
Lazear, E.P., 2000b. Performance pay and productivity. Am. Econ. Rev. 90 (0), 1346–1361.
Lazear, E.P., 2001. Educational production. Q. J. Econ. 116 (3), 777–803.
Lazear, E.P., Malmendier, U., Weber, R.A., 2012. Sorting in experiments with application to social preferences. Am. Econ. J. 4 (1), 136–163.
Lee, G.K., Cole, R.E., 2003. From a firm-based to a community-based model of knowledge creation: the case of the Linux kernel development. Organ. Sci. 14 (6), 633–649.
Lee, S., Moisa, N., Weiss, M., 2003. Open source as a signalling device-an economic analysis. Working Paper Series: Finance & Accounting. Johann Wolfgang Goethe-Universität Frankfurt am Main.
Lerner, J., Tirole, J., 2002. Some simple economics of open source. J. Ind. Econ. 50 (2), 197–234.
Levine, S.S., Prietula, M.J., 2013. Open collaboration for innovation: principles and performance. Organ. Sci. 25 (5), 1414–1433.
Levy, S., 1984. Hackers: Heroes of the Computer Revolution. Anchor Press/Doubleday, Garden City, NY.
Majchrzak, A., Malhotra, A., 2020. Unleashing the Crowd: Collaborative Solutions to Wicked Business and Societal Problems. Palgrave Macmillan, Cham, Switzerland.
Mallapragada, G., Grewal, R., Lilien, G., 2012. User-generated open source products: founder's social capital and time to product release. Mark. Sci. 31 (3), 474–492.
Marlow, J., Dabbish, L., 2013. Activity traces and signals in software developer recruitment and hiring. In: Proceedings Conference on Computer Supported Cooperative Work. (San Antonio, Texas, USA), pp. 145–156.
Marlow, J., Dabbish, L., Herbsleb, J., 2013. Impression formation in online peer production: activity traces and personal profiles in GitHub. In: Proceedings Conference on Computer Supported Cooperative Work. (San Antonio, Texas, USA), pp. 117–128.
Maruping, L.M., Daniel, S.L., Cataldo, M., 2019. Developer centrality and the impact of value congruence and incongruence on commitment and code contribution activity in open source software communities. MIS Q. 43 (3), 951–976.
Mateos-Garcia, J., Steinmueller, W.E., 2008. The institutions of open source software: examining the Debian community. Inf. Econ. Policy 20 (4), 333–344.
McDonald, N., Goggins, S., 2013. Performance and participation in open source software on GitHub. In: Proceedings CHI'13 Extended Abstracts on Human Factors in Computing Systems. (Paris, France), pp. 139–144.
Merton, R.K., 1968. The Matthew effect in science: the reward and communication systems of science are considered. Science 159 (3810), 56–63.
Mingers, J., Walsham, G., 2010. Toward ethical information systems: the contribution of discourse ethics. MIS Q. 34 (4), 833–854.
Mockus, A., Fielding, R.T., Herbsleb, J.D., 2002. Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol. 11 (3), 309–346.
Moon, J.Y., Sproull, L.S., 2008. The role of feedback in managing the Internet-based volunteer work force. Inf. Syst. Res. 19 (4), 494–515.
Nelson, M., Sen, R., Subramaniam, C., 2006. Understanding open source software: a research classification framework. Commun. AIS 17 (1), 266–287.
Nonnecke, B., Preece, J., 2003. Silent participants: getting to know lurkers better. From usenet to CoWebs. Springer, London, pp. 110–132.
Ollo-Lopez, A., Bayo-Moriones, A., Larraza-Kintana, M., 2010. The relationship between new work practices and employee effort. J. Ind. Relat. 52 (2), 219–235.
O'Mahony, S., Bechky, B.A., 2008. Boundary organizations: enabling collaboration among unexpected allies. Adm. Sci. Q. 53 (3), 422–459.
O'Mahony, S., Ferraro, F., 2007. The emergence of governance in an open source community. Acad. Manage. J. 50 (5), 1079–1106.
Piezunka, H., Dahlander, L., 2015. Distant search, narrow attention: how crowding alters organizations' filtering of suggestions in crowdsourcing. Acad. Manage. J. 58 (3), 856–880.
Piezunka, H., Dahlander, L., 2019. Idea rejected, tie formed: organizations' feedback on crowdsourced ideas. Acad. Manage. J. 62 (2), 503–530.
Puranam, P., Alexy, O., Reitzig, M., 2014. What's "new" about new forms of organizing? Acad. Manage. Rev. 39 (2), 162–180.
Raveendran, M., Puranam, P., Warglien, M., 2021. Division of labor through self-selection. Organ. Sci. In press.
Raymond, E., 1999. The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly & Associates, Sebastopolous, CA.
Riedl, C., Seidel, V.P., 2018. Learning from mixed signals in online innovation communities. Organ. Sci. 29 (6), 1010–1032.
Roberts, J.A., Hann, I., Slaughter, S.A., 2006. Understanding the motivations, participation, and performance of open source software developers: a longitudinal study of the Apache projects. Manage. Sci. 52 (7), 984–999.
Sanders, J., 1998. Linux, open source, and software's future. IEEE Softw. 15 (5), 88–91.
Santos, C., Kuk, G., Kon, F., Pearson, J., 2013. The attraction of contributors in free and open source software projects. J. Strat. Inf. Syst. 22 (1), 26–45.
Schøne, P., 2009. New technologies, new work practices and the age structure of the workers. J. Popul. Econ. 22 (3), 803–826.
Sen, R., Subramaniam, C., Nelson, M.L., 2008. Determinants of the choice of open source software license. J. Manage. Inf. Syst. 25 (3), 207–240.
Shah, S.K., 2006. Motivation, governance, and the viability of hybrid forms. Manage. Sci. 52 (7), 1000–1014.
Siggelkow, N., Rivkin, J.W., 2009. Hiding the evidence of valid theories: how coupled search processes obscure performance differences among organizations. Adm. Sci. Q. 54 (4), 602–634.
Simon, H.A., 1951. A formal theory of the employment relationship. Econometrica 19 (3), 293–305.
Spaeth, S., von Krogh, G., He, F., 2015. Research note—perceived firm attributes and intrinsic motivation in sponsored open source software projects. Inf. Syst. Res. 26 (1), 224–237.
Stallman, R., 1999. The GNU operating system and the free software movement. In: DiBona, C., Ockman, S., Stone, M (Eds.), Open Sources: Voices from the Open Source Revolution. O'Reilly & Associates, Sebastopol, CA, pp. 53–70.
Staw, B.M., 1981. The escalation of commitment to a course of action. Acad. Manage. Rev. 6 (4), 577–587.
Stewart, K.J., Ammeter, A.P., Maruping, L.M., 2006. Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects. Inf. Syst. Res. 17 (2), 126–144.
Stewart, K.J., Gosain, S., 2006. The impact of ideology on effectiveness in open source software development teams. MIS Q. 30 (2), 291–314.
Subramaniam, C., Sen, R., Nelson, M.L., 2009. Determinants of open source software project success: a longitudinal study. Decis. Support Syst. 46 (2), 576–585.
Tsay, J., Dabbish, L., Herbsleb, J., 2013. Social media in transparent work environments. In: Proceedings 6th International Workshop on Cooperative and Human Aspects of Software Engineering. (San Francisco, USA), pp. 65–72.
Tsay, J., Dabbish, L., Herbsleb, J., 2014a. Let's talk about it: evaluating contributions through discussion in GitHub. In: Proceedings 22nd International Symposium on Foundations of Software Engineering. (ACM, Hong Kong, China), pp. 144–154.
Tsay, J., Dabbish, L., Herbsleb, J., 2014b. Influence of social and technical factors for evaluating contribution in GitHub. In: Proceedings 36th International Conference on Software Engineering. (Hyderabad, India), pp. 356–366.
Vasilescu, B., Filkov, V., Serebrenik, A., 2013. StackOverflow and GitHub: associations between software development and crowdsourced knowledge. In: Proceedings International Conference on Social Computing. (Washington DC, USA), pp. 188–195.
von Hippel, E., von Krogh, G., 2003. Open source software and the "private-collective" innovation model: Issues for organization science. Organ. Sci. 14 (2), 209–223.
von Krogh, G., Haefliger, S., Spaeth, S., Wallin, M.W., 2012. Carrots and rainbows: Motivation and social practice in open source software development. MIS Q. 36 (2), 649–676.
von Krogh, G., Spaeth, S., Lakhani, K.R., 2003. Community, joining, and specialization in open source software innovation: a case study. Res. Policy 32 (7), 1217–1241.
Wasko, M.M., Faraj, S., 2000. "It is what one does": why people participate and help others in electronic communities of practice. J. Strat. Inf. Syst. 9 (2-3), 155–173.
Wasko, M.M., Faraj, S., 2005. Why should I share? Examining social capital and knowledge contribution in electronic networks of practice. MIS Q. 29 (1), 35–57.
West, J., O'Mahony, S., 2008. The role of participation architecture in growing sponsored open source communities. Ind. Innov. 15 (2), 145–168.
Wooldridge, J., 2010. Econometric Analysis of Cross-Section and Panel Data, second ed. MIT Press, Cambridge, MA.
Wooten, J.O., Ulrich, K.T., 2017. Idea generation and the role of feedback: Evidence from field experiments with innovation tournaments. Prod. Oper. Manag. 26 (1), 80–99.
Xu, B., Jones, D.R., Shao, B., 2009. Volunteers' involvement in online community based software development. Inf. Manage. 46 (3), 151–158.
Zhang, C., Hahn, J., De, P., 2013. Research note—Continued participation in online innovation communities: does community response matter equally for everyone? Inf. Syst. Res. 24 (4), 1112–1130.
Zhu, H., Zhang, A., He, J., Kraut, R.E., Kittur, A., 2013. Effects of peer feedback on contribution: a field experiment in Wikipedia. In: Proceedings Conference on Human Factors in Computing Systems. (Paris, France), pp. 2253–2262.