Sharon Belenzon and Mark Schankerman

Motivation and sorting of human capital in open innovation

Article (Accepted version)
(Refereed)

Motivation and Sorting of Human Capital in Open Innovation

Sharon Belenzon and Mark Schankerman

May 15, 2014

Abstract

This paper studies how business models can be designed to tap effectively into open innovation labor markets with heterogeneously motivated workers. Using data on open source software, we show that motivations are diverse, and demonstrate how managers can strategically influence the flow of code contributions and their impact on project performance. Unlike previous literature using survey data, we exploit the observed pattern of project membership and code contributions – the "revealed preference" of developers—to infer the motivations driving their decision to contribute. Developers strongly sort along key dimensions of the business model chosen by project managers, especially the degree of openness of the project license. The results indicate an important role for intrinsic motivation, reputation, and labor market signaling, and a more limited role for reciprocity.

Keywords: strategic human capital, sorting, motivations, open innovation, open source, intellectual property rights

1. Introduction

Strategy research explores how heterogeneity in firm resources can lead to differential performance. Within this view, employees are among the most important types of rent-generating resources, due to scarcity, specialization, and tacit knowledge. Building on resource-based theory, the human capital management literature has suggested policies designed to increase employee motivation in the face of several important challenges. Unlike physical assets, workers can freely leave the firm (Coff, 1997; Coff and Kryscynski, 2011), their contribution to performance is hard to observe, and their actions may be driven by diverse motivations, especially in knowledge-intensive activities (Sauermann and Cohen, 2010).

This paper focuses on an understudied link between business strategy and human capital – namely, that the ability to tap into a pool of talent is strongly related to the specific business model chosen by managers. Our paper studies this link in the context of open source software (OSS). In OSS, source code contributors are typically unpaid and the source code is available for anyone to use and modify under project-specific conditions. This innovation model allows us to isolate the impact a specific business model has on the composition and quality of human capital. The main feature we focus on in this paper is the degree of property rights protection that is chosen by the project manager. OSS plays an important role in the commercialization strategies of software firms. Companies often comingle the development of open source and proprietary products, where nearly 40% of software firms participate in both proprietary and open source development and more than 30% of these firms devote at least half of their computer developers' time to OSS (Lerner and Schankerman, 2010). Importantly in our context, evidence suggests that firms tend to adopt a *diverse* set of business models that involve varying degrees of property rights, ranging from fully proprietary to more open intellectual property regimes (Lerner and Schankerman, 2010; Fosfuri et al., 2008). Our focus in this paper is on the labor market-implications of selecting varying degrees of property rights protection and underscores an important and understudied tradeoff between stronger property rights and sorting of intrinsically motivated workers.

We highlight five features that make OSS particularly interesting for strategic human capital research. First, OSS is based on team production, rendering the level and quality of effort difficult to observe. In the absence of direct monitoring, aligning incentives between the developer and the project manager plays a major role. Second, OSS participants are not formally employed by the organization with which they are affiliated, which makes informal relationships, one of the core subjects of the human capital literature, a central issue. Third, the ease of worker mobility manifests itself most pervasively in OSS, which raises the question of how organizations can generate a successful human capital strategy in an extremely fluid labor market with no formal contracts to bind workers. Fourth, OSS communities are known for their diverse motivations, which range from highly ideological (intrinsic) to more careerist (extrinsic) concerns. In this setting, understanding worker motivations can help firms design better employment contracts that compensate workers on the basis of what they care about, and allow workers to sort into organizations characterized by resonant goals. Fifth, OSS is an extremely decentralized system of organizing innovation activity – workers have full autonomy over the tasks they perform. A recent paper by Gambardella et al. (2013) argues that providing workers with more autonomy can create a loss for the firm due to misaligned objectives, but at the same time can increase private benefits for workers – mitigating agency problems. This argument suggests that under some conditions, firms may choose to create a decentralized system – similar to what we observe in OSS. Our paper contributes to this discussion by studying how to organize inventive activity in the presence of heterogeneous worker motivations. In our setting, all workers are autonomous, but they vary in how much they value this autonomy relative to other sources of private benefits. At the same time, firms differ in their costs of attracting and motivating workers through forgone property rights. We argue that the matching driven by heterogeneity – at the level of the worker and the firm (business model) – is understudied and should be a more central theme at the intersection between business strategy and strategic human capital domains.

Our empirical strategy is to utilize *revealed preferences*, as captured by the observed pattern of project membership and source code contributions, to quantify how reputation, reciprocity, utility, and intrinsic motivations drive open source innovation. Our empirical analysis of source code contributions is based on a large-scale data set with detailed information on the contributing and receiving OSS projects. Each contribution includes a dyad with a contributing project, where the contributor is a registered member, and a receiving project to which the source code is submitted. The distinction between contributing and receiving projects is central to our empirical analysis, because we seek to establish whether developers affiliated with certain types of contributing projects systematically target certain kinds of receiving projects. The key variation comes from projects' variation in the degree to which their licenses are "open." Though open, unrestricted access was the original driving force behind the "free software" movement (Raymond, 2001), many projects now incorporate OSS contributions under a variety of licenses that allow the source code to be used in proprietary ways that limit terms of use. For clarity, we refer to the latter as "closed" projects.

To study sorting, we exploit project characteristics such as license type, size, programming language, operating system, and intended audience. We study the empirical determinants of contributions by focusing on four distinct groups of developers, whose profile we infer based on to the types of open source licenses that govern the projects with which they are affiliated: open, closed, mixed, and anonymous. We investigate how the pattern of contributions from each developer type varies across the characteristics of the contributing-receiving dyads. The key innovation in our approach is that we exploit the observed pattern of contributions—the "revealed preferences" of developers—to infer the underlying motivations, unlike earlier work that is based on survey information. Our econometric approach is to aggregate source code contributions into cells defined by a set of detailed characteristics of the contributing developers and receiving projects, and then to use these cells as the observations in the estimation procedure. We then show how the likelihood of source code contribution varies with the motivation of the contributing developer, and the potential (intrinsic/extrinsic) reward associated with matching the contributor to specific bundles of receiving project characteristics. We find that the heterogeneity of motivations in the open source community leads to systematic sorting of developers on key dimensions of business model choices managers make, including the openness of project license, project size, intended audience and corporate sponsorship.

Can sorting be a source of competitive advantage? Gottschalg and Zollo (2007) address this question by introducing the concept of interest alignment into competitive strategy research. In their view, the answer depends on whether firms differ in the costs or benefits of implementing profitable sorting between workers and firms. In the context of OSS, the openness of the license affects the flow of contributions from different types of developers, and in particular, the mix of intrinsically motivated versus commercially oriented developers. The costs of a more open license is weaker property rights over the resulting source code. The manager's choice regarding license type will depend on the balance between these two forces. Competitive advantage can arise only if the balance between these costs and benefits varies across firms in ways that are difficult to imitate. Evidence by Schankerman and Lerner (2010) suggest such variation exists. They show that the degree to which firms comingle the development of open source and proprietary software varies substantially, as do the specific types of software license modes and business models they use to do so. For example, they find that companies that develop proprietary software are more likely to engage in exports (where protection is more important) than firms that focus on open source. Fosfuri et al. (2008) also present evidence suggesting that firms differ in their ability to profit from open source, and that this ability is strongly related to heterogeneity in complementary downstream capabilities.

In summary, our findings illustrate that business model design can have important labor market implications, especially when intrinsically motivated developers are an important part of the labor force. We provide econometric evidence that the heterogeneity of motivations is closely linked to the optimal design of business models for knowledge workers in open innovation. Specifically, we demonstrate and quantify the presence of a tradeoff between stronger property rights project managers choose and the flow of contributions the project receives from the most productive, intrinsically motivated, developers.

2. OSS as a Window into Open Innovation Systems

The literature discusses open innovation in two distinct ways. The first is an innovation model that optimally integrates internal and external markets. Whereas classical transaction cost economics focuses on the allocation of activity between firms and markets, a new paradigm has emerged to emphasize the integration of firms and markets. This open innovation paradigm refers to the "use of external knowledge ideas as well as internal ideas, and internal and external paths to markets, as firms look to advance their technology" (Chesbrough, 2003, p. 1). Technology markets lie at the heart of this literature because they are key channels by which firms access critical technological knowledge and inputs (Arora et al., 2001). For such markets to function effectively, transacting parties need to have strong patent rights. Patents facilitate technology exchange by reducing expropriation risk, acting as bargaining chips that enhance firms' ability to contract over bundles of technologies, and by enforcing patent rights without reliance on costly litigation. In particular, patents help ensure knowledge access is limited to those firms that either create it, or to other transacting firms that create technologies that could be exchanged through market transactions.

A second strand of the literature views openness from a very different lens, characterizing it by the proprietary nature of the intellectual property regime and the collectiveness of innovation (Baldwin and von Hippel, 2011). This literature examines whether innovation can be sustained in the absence of patents. This collective innovation model is aligned with Chesbrough's notion of openness in emphasizing the combination of internal and external research, but this view treats patents as obstacles, not drivers, of technical progress. In this perspective, private returns to R&D are realized through product market competition in the form of improved products and services, but the innovation market remains open with firms drawing on and contributing to a growing pool of common knowledge. Free riding is not presumed to be important, and for this reason, patents should not have any positive effect on R&D incentives.

OSS is an environment where both types of openness coexist. Open source builds on the successful combination of internal and external code contributions, consistent with Chesbrough's view. At the same time, developers receive no formal property rights over their code contributions, consistent with the collective innovation model. However, the open source process also exhibits important "closed" elements. Whereas some projects attract many external contributions to complement their internally developed code, the vast majority of projects fail to attract any external contributions. Moreover, open source projects vary widely in the nature of the associated intellectual proprietary rights. Whereas some projects restrict the commercial use of code, others are much more flexible. We emphasize that this choice of license type, along with other characteristics we will study, is a key dimension of the business model managers must choose.

A central concern for policy makers and firms alike is the sustainability of open innovation systems and the different domains where they can successfully flourish (Cohen, 2005). For open systems to be sustainable, free-riding must be sufficiently mitigated: in the present context, the marginal private rewards from individually contributing a source code must be larger than those from not contributing. von Hippel and Krogh (2002) emphasize that what is required for free riding to be mitigated is "selective incentives" – private rewards that accrue only to contributing developers. To illustrate this point, suppose that developers have a single motivation – improving their employment opportunities. By contributing to open source projects developers, can signal their quality to potential employers, such as software firms. If those firms can easily observe the effort and quality of contributions, free riding would not be a concern, because selective incentives would be large. Developers that do not contribute or those that make low-quality contributions would not gain any private rewards, which means that even in the case of a single motivation, free-riding may not be fatal.

However, in some circumstances, firms cannot easily observe the level and quality of effort. In such cases, the heterogeneity of worker motivations and of the business models firms adopt to commercialize their products become central for sustainability. Developers that seek to improve their technical skills, for example, would contribute even when firms do not observe their effort and quality. Thus, with non-pecuniary incentives, contribution is now more likely than in the case in which career concern is the only motivation. Further, if the "free-software" ideology also drives developers, contributions would be even more robust. If, for instance, learning opportunities are exhausted as developers become more experienced, the free-software ideology would ensure continued contributions. But for continued contributions driven by the free-software ideology, firms need to offer “motivated” developers the opportunity to work on open projects. Thus the open system is more likely to be sustainable when both workers have heterogeneous motivations *and* firms adopt business models that effectively exploit this worker heterogeneity. In our empirical work, we show that diverse motivations lead developers to sort systematically on project characteristics and that this sorting can be exploited by firms strategically to organize their projects to maximize the flow of contributions and their impact on project performance.

3. Heterogenous Motivations and Sorting

Motivations fall into two broad classes: extrinsic and intrinsic. The concept of extrinsic motivation was first introduced by Skinner (1953) and further developed by Deci (1972), who defined extrinsically motivated behaviors as those that are seen by actors as means to an end. When thusly motivated, actions are performed merely to attain some separable outcome to which external rewards are attached (e.g., status, approval, or monetary compensation). This is what most people in economics and management would associate with "incentives". In the absence of a commensurably meaningful reward, the action would not be performed. By contrast, an individual is considered intrinsically motivated to act if she "performs an activity for no apparent reward except the activity itself" (Deci, 1972).

Intrinsic motivation has been especially important for the study of innovation. Several papers emphasize the role of intrinsic motivations in supporting innovating activities, and specifically that more science-oriented jobs typically involve some degree of autonomy (Stern, 2004; Sauermann and Cohen, 2010). Our paper complements this research in two ways. First, it demonstrates the presence of heterogeneous motivations in OSS where "quasi-firms" and projects operate in a labor market environment without formal contracts and highly mobile workforce (whereas most previous papers in the area focus on motivations within discrete firms with formal labor contracts). Second, our paper underscores the importance of sorting – the matching between worker preferences and project characteristics (business model).[1]

Studies of motivations in OSS have proposed four main explanations. First, developers may be intrinsically motivated to contribute by their strong identification with the "ideology" underlying the open source movement (Raymond, 2001). The original open source license that embodies this view is the general purpose license (GPL), which requires that the source code and any subsequent source code that builds on it or embodies it must remain open source. Second, source code contribution and active participation in the OSS community may enhance the developer's reputation via peer recognition (Raymond, 2001) or commercial rewards in the labor market (Lerner and Tirole, 2001, 2002; Johnson, 2002). Third, developers may expect later gains from reciprocal contributions from projects to which they have previously contributed (Lakhani and von Hippel, 2003). Fourth, developers may also enjoy (get utility value from) participation (Shah, 2006). Within each of these explanations exists some varying degree of both intrinsic and extrinsic motivations, which have been treated empirically in a number of studies that are based on survey information within an open source setting. But none of these studies developed a link between the choice of business model and the ability of firms to tap into such heterogenous labor markets. Lakhani and Wolf (2005), using survey instruments on a sample of OSS developers, find that intrinsic motivation is the strongest driver of contributions. A related paper is Roberts et al. (2006), who find a role for both intrinsic and extrinsic motivations, among other interesting findings.

4. Development of Hypotheses

A number of survey-based papers have emphasized the importance of non-pecuniary motivations to contribute to open source projects. These motivations include ideology, reciprocity, enjoyment, and technical learning (Lakhani and von Hippel, 2003; Shah, 2006). Based on this literature, we identify three types of developers as implied by the expected relative importance they attach to intrinsic motivation when making source code contributions: *open*, *closed*, and *mixed*. We assume that developers reveal themselves as being intrinsically motivated if they are members only of projects that have open licenses, because these projects adhere more closely to the original OSS philosophy. Conversely, we assume developers reveal a preference for extrinsic motivation if they belong only to projects with closed license types. We consider as *mixed* those developers for which no clear evidence exists on whether their motivation is primarily intrinsic or extrinsic.

Building on prior literature, we identify four classes of motivations, from highly intrinsic to highly extrinsic: *Pure intrinsic motivation, Utility/Learning, Reciprocity,* and *Reputation.* The literature and available

[1]Scholarship in economics has historically not embraced the concept of intrinsic motivation, though in recent years it has begun to garner more attention (e.g., Bénabou and Tirole, 2003; Besley and Ghatak, 2005).

small-sample survey evidence suggest a role for each of these in open source development (e.g., Haruvy et al.; Lakhani and von Hippel, 2003; Hertel et al., 2003; Lerner and Tirole, 2002). Of the four motivation types, reputation relies most directly on extrinsic mechanisms: the very notion of "reputation" requires an external agent to form an opinion of the subject. At the other extreme, pure intrinsic motivation is simply defined as lacking extrinsic dimensions. On the other hand, reciprocity and utility/learning are likely to lie along the continuum, because both share features of internal and external motivations. Our hypotheses aim to assess the relative importance of these various motivations in driving OSS contributions, while acknowledging that agents may be driven by more than one motivation type. We proceed by discussing the empirical implication of each motivation class, and we suggest testable hypotheses.

Our analysis is an effort to document, within a set of OSS developers, systematic patterns of behavior that are consistent with the predictions of intrinsic and extrinsic motivation theory. We examine how developers respond to different choices of project characteristics to infer underlying motivations, as well as to study the implications of different business model choices by project managers (e.g., property rights, intended audience, and project size) for the flow of source code contributions and the resulting project performance.

Pure intrinsic motivation. As discussed earlier, a salient feature of OSS is that many developers have a strong ideological preference for keeping source code fully open. In economic terms, this feature means that for a developer affiliated only with open projects, the utility derived from making a source code contribution is larger if it is for an open project. In the extreme case, this utility would be zero if the contributions were made to a project with any other license type. In psychological terms, purely motivated developers should achieve greater levels of autonomy through acting in accord with self-endorsed values, needs, and intentions, as well as a stronger sense of ideological *relatedness* to the relevant OSS community. Because reputation is strongly associated with extrinsic motivation, anonymous developers, for whom reputation plays no role, are best viewed as intrinsically motivated. For that reason, we expect their pattern of contributions to be similar to that of open developers. This discussion leads to our first hypothesis:

Hypothesis 1a (intrinsic motivation, open/anonymous developers): Open (closed) projects should receive more (fewer) contributions from anonymous and open developers.

This aspect of the intrinsic motivation hypothesis predicts positive (but not necessarily exclusive) sorting of open developers to open-license projects. That is, more intrinsically motivated developers (those more likely to be anonymous, or open developers who are members of projects with highly open licenses) would be more likely to contribute to projects with highly open licenses.

The debate between advocates of open source and proprietary software has been polarized, with strident criticism from both sides of the divide. In this context, some developers might be ideologically motivated against contributing to highly open projects, whose licenses may be viewed as "anti-property rights."[2] To the extent that this view is widely held, the ideological motivation may also induce sorting by closed developers. This type of sorting should be considered a form of intrinsic motivation, because it is ideologically based. This discussion leads to the following:

Hypothesis 1b (intrinsic motivation, closed developers): Closed (open) projects should receive more (fewer) contributions from closed developers.

Reputation. The reputation-related motivations in OSS communities have been well documented (e.g., Hertel et al., 2003). These benefits are much more starkly extrinsic, because the way in which status and respect from the other members of a community may motivate contributors is clear. However, their importance in comparison to other types of motivation is subject to debate. Lerner and Tirole (2002) argue that developers improve their labor market prospects by signaling their quality through participation in open source projects. These signaling benefits are likely to be greater: (1) when the project to which they contribute is larger and more visible (Johnson, 2002); (2) when the project reveals the outcome of the contribution, for example, by being accepted by the project manager; (3) when the project is sponsored by commercial firms; and (4) when the project is aimed at developer-users rather than end users. The prediction of the commercial reputation (labor market signaling) hypothesis is that closed developers should sort positively on these dimensions. Importantly, because reputational benefits clearly should not matter to anonymous contributors, this group can serve as a benchmark against which reputation effects are evaluated.[3]

The second type of reputation gain is peer recognition, which is unrelated to labor market payoffs. Peer recognition should also be greater for larger projects, those that reveal whether the project manager has accepted the code contribution (public resolution), and those aimed at programming tools used by other developers rather than by end users. Moreover, among non-anonymous developers, we expect reputation to be less important for contributors who sort (primarily) into open projects, because intrinsic motivation plays a more important role for them. By contrast, extrinsically motivated (closed) developers should be more attracted to more visible projects, where their gains from reputation are likely to be greater. This discussion leads to the following set of hypotheses:

Hypothesis 2a (reputation, project size): All developer types, except anonymous, should be more likely

[2]The highly open GPL license is also known by some open source advocates as "copyleft" to emphasize this anti-property-rights perspective.

[3]For recent work in open innovation outside the OSS which underscores the reputational effects for innovation activity, see Boudreau et al. (2011) and Jeppesen and Lakhani (2010).

to contribute to larger projects.

Hypothesis 2b (reputation, visible contribution outcome): All developer types, except anonymous, should be more likely to contribute to projects that reveal outcomes of contributions (public resolution). Anonymous developers should either be unaffected by public resolution, or be negatively affected if public resolution crowds out their intrinsic (ideology) motivation.

As proposed by Lerner and Tirole (2002), commercial sponsorship of a project can increase effectiveness of labor market signaling by developers, which should make such projects more attractive to extrinsically motivated developers. Thus:

Hypothesis 2c (reputation, corporate sponsorship): Closed developers should be more likely to contribute to projects sponsored by corporations. However, anonymous and open developers should either be unaffected by corporate sponsorship, or be negatively affected, if corporate sponsorship crowds out their intrinsic motivation (ideology).

Finally, reputation gains are likely to be greater when the developer's peer group is more able to understand the technical merit of her contributions, both for reputation in the labor market and for peer recognition. To the extent that reputation is valuable to extrinsically motivated developers, we expect these gains to be larger when the intended audience is other software developers rather than end users. This discussion leads to the following:

Hypothesis 2d (reputation, intended audience): Both open and closed developers should be more likely to contribute to projects that are intended to be used by developers (as opposed to end users). Anonymous developers should not systematically sort on this dimension.

The above features, which affect reputation-driven developers, are part of the business model the project manager chooses. For example, the manager can choose project size by can determining the number of members she wishes to admit. Moreover, the choice between niche and broad positioning of the project will affect project size. Hypothesis 2a suggests that more niche projects would attract developers that are less driven by reputation concerns, making anonymous developers the ideal candidates for such projects. If anonymous developers are also intrinsically motivated (Hypothesis 1a), the project manager should strongly consider setting a highly-open license type.

Reciprocity. Survey evidence clearly points to the importance of reciprocity in driving code contribution. Shah (2006) finds that one of the most important reasons for developers to contribute was a sense of reciprocity. Aside from deriving satisfaction from developing a source code, contributors felt that helping others in the community was important because they had benefited from others' contributions. Similarly, Lakhani and von Hippel (2003) find that reciprocity is the most important reason contributors cite for posting answers on Usenet groups. Contributors either repay the benefits they received or contribute in expectation of benefiting from the community in the future. This discussion suggests the following:

Hypothesis 3a (reciprocity, prevalence): Contributions from members of project i to project j in year t should be more likely when members of project j have contributed to project i at some point prior to year t.

The earliest proponents of open source emphasized the role that reciprocity - sometimes called "gift culture" - plays in sustaining incentives for innovation. Their argument is that developers who are members of a project may make contributions in direct response to, or anticipation of, contributions other developers make to their project. Importantly, reciprocity is typically viewed as a self-sustaining mechanism largely, if not wholly, divorced from commercial considerations (e.g., Raymond, 2001). Therefore, we expect the following:

Hypothesis 3b (reciprocity, non-commercial): Reciprocity should be more important for projects that have highly open licenses.

At the same time, we have two reasons to suspect reciprocity is more important for commercial projects. First, reciprocity can serve as an informal payback mechanism for developers driven by extrinsic rewards. Second, in our context, economic theory shows reciprocity can be sustained if contributing developers can detect and effectively punish developers who deviate from reciprocity strategies. As we show in this paper, developers that contribute to closed projects are typically concerned about their reputation, which makes their identity more readily known and makes deviation easier to spot and punish. This discussion suggests the following:

Hypothesis 3c (reciprocity, commercial): Reciprocity should be more important for projects that have closed licenses.

5. Data

The data are taken from SourceForge.net, the largest web host for OSS projects. SourceForge provides a publicly accessible platform, introduced in 1999, in which developers interact during the software development process. We developed specialized software algorithms that accessed each project registered on SourceForge over the period 1999-2010, and extracted all available information about the project, the participating developers, and their software source code contributions. The online Data Appendix provides a detailed description of the data collection, as well as a number of consistency checks.

A *contribution* is defined as any contributed source code that aims to advance the software regardless of whether the project manager ultimately accepts the submission. This definition is broader than that used in past work that typically look at source code contributions that have been incorporated in the software (e.g., Giuri et. al, 2010; Lerner et. al, 2006). We prefer the broader definition because of our interest in studying how developers direct their programming efforts to different projects depending of their type and project characteristics. We define *external contributions* as source code submitted by developers who are not registered members of the receiving project, and *internal contributions* as source code submitted by developers to projects of which they are members. In the sample, 39% of contributions are internal. The remaining 61% are external, coming from developers who are formally affiliated with other projects (but not the focal project), are not members of any project, or do not reveal their identity when making their contribution (anonymous developers). We focus the analysis of sorting behavior on the pattern of these external contributions.

5.1. Main variables

The key variables in the empirical analysis are as follows:

Project License Type: The most important project characteristic we consider is license type. Each project is governed by a set of rules that define the terms of use of the software developed by its members and other participants. These terms of use are defined by the project license, which focuses mainly on the extent to which commercial use is allowed. Licenses that constrain such use more severely are referred to in the literature on open source as "more restrictive." We label the projects governed by these licenses as "open," because the impact of these restrictive licenses is to keep software free, in the spirit of "open software." Two main features define the restrictiveness of a license: (1) the extent to which the source code and any of its modifications can be subsequently embodied in commercial software and (2) whether modifications to the source code have to remain open source (i.e., the binary source code must remain open and accessible).[4] The projects in our data cover about 44 license types. Using the description of each license type (http://www.opensource.org/licenses), we classify licenses into three categories:

1. *Highly Open (HO):* This type includes the General Purpose License (GPL). It requires that any file, regardless of source code origin, that is combined under certain circumstances with a file under GPL

[4]For a discussion of different license types and their restrictions, see Lerner and Tirole (2002).

must be licensed under GPL. This license type is regarded as ideologically closest to the original idea behind the "free software" movement, and its objective is to preserve a fully open software commons and limit commercial gains from software development to the maximum possible extent.

2. *Open (O)*: The license requires that modified versions of the program can only be distributed if the source code remains open, but it can be used commercially. The license conditions carry no restrictions on the modifications and extensions of the source code, provided it remains open source. Examples include Lesser GPL, Common Public License, and Sun Public License.

3. *Closed (C):* The license allows modifications and extensions of the open source code to be integrated into commercial software, and these do not have to remain open source. Examples include BSD, Python, and MIT.

In our sample, among projects that receive at least one code contribution 60% operate under highly open licenses and 25% under closed licenses.[5]

Developer Types: We assign each developer a "type" based on the types of projects to which she belongs as a registered member. Project membership is an important decision that ties developers to specific projects. This tie is likely to impact the developer's reputation in the OSS community, and beyond, because her association with projects becomes public information that can reveal her preferences to outsiders and link her personal reputation to the future performance of the project. We classify a developer as *open* if all of the projects to which she belongs operate under highly open licenses, as defined above. We define a developer as *closed* if all of the projects to which she belongs operate under closed licenses. All other developers who are members of projects are classified as *mixed*.[6] In addition, we use two other categories: "Anonymous" developers are those who contribute without revealing their identity to the receiving project manager or members. "Non-members" are contributors who are not registered as members of any project.

Our sample includes 149,956 unique developers: 68% have no affiliation to any project, 15% are open, 9% are closed, and the remaining are mixed. Of the total sample of developers, 15% make at least one contribution to projects with which they are not affiliated ("external contributions") over the sample period.

Size of Project: The size of the project is defined by the number of developers registered as formal members, including the project manager. Size is an important measure in our analysis because it is our key

[5]For 7% of the projects, we observe multiple licenses. In these cases, we classify the project as highly open if at least one of its licenses is highly open, and as closed if all of its licenses are closed. The results reported in the paper are robust to alternative assumptions, such as classifying projects as highly open only if all of their licenses are highly open, or classifying projects as closed only if the majority of their licenses are closed. The remaining projects are classified as open.

[6]The majority of developers belong to very few projects—the mean number of projects of which a member developer belongs is 1.4 (median is 1, 99th percentile is 7).

test of Hypothesis 2a on the reputation motive. Importantly, the project manager can potentially influence the size of the project by allowing more or fewer developers to become formal members. We show that this decision has important implications for sorting.

The median project has one member and the average is 4.1. The distribution is sharply skewed, however—the project at the 90th percentile of the size distribution has 10 members (99th percentile has 37 members). Larger projects receive more (external) contributions than small projects. Conditional on receiving at least one external contribution, projects above the median size receive an average of 28.5 contributions, compared to only 8.8 for below-median-size projects.

Intended Audience: SourceForge identifies the intended audience for each registered software project from among 19 groups. We aggregate these groups into five categories for the empirical analysis: Developers (programming tools), End Users, System Administrators, Mixed (of the preceding three), and Other.[7] About 30% of projects receiving contributions are developer-oriented and 18% target end users.

In the econometric analysis, we also control in all regressions for project programming language and operating system. The online Data Appendix provides details on these controls.

6. Econometric Specification

Our primary objective is to estimate the effect of project characteristics on the pattern of source code contributions. For this estimation, we first aggregate contributions into cells, where each cell is defined by a set of characteristics of the developer type and receiving projects. These cells become the observations in the estimation procedure. The empirical task is to relate the number of contributions between different cells to the characteristics of the developers and projects defining those cells. Because the number of contributions is an integer, we use an econometric model for count data. We adopt a Negative Binomial specification:

$$Y_{c,r} = \exp(\alpha X_r + \lambda X_c X_r + \upsilon_{cr}) \tag{1}$$

where $Y_{c,r}$ denotes the number of contributions from projects in cell c to projects in cell r, X_c is a vector of characteristics of the contributing project c (including the developer type), X_r is a vector of characteristics of the receiving project r, and we assume the negative binomial error is conditionally independent of the characteristics in (X_c, X_r), $E(\upsilon_{cr} \mid X_c, X_r) = 0$. The model is estimated by maximum likelihood.[8]

[7]The End User category includes end users/desktop and advanced end users. The "Other" category, which accounts for 4% of projects, includes mostly aerospace, education, science/research, and healthcare.

[8]The alternative, Poisson model imposes the strong restriction that the conditional mean and variance of Y_{cr} are equal. The Negative Binomial model allows for "overdispersion" of the form $Var(Y_{c,r}) = E(Y_{c,r}) + \phi[E(Y_{c,r})]^2$ and estimates the overdispersion parameter ϕ along with the other parameters. Our estimates easily reject the Poisson case $\phi = 0$.

As equation 1 shows, the regressions incorporate additive controls for project characteristics that are designed to capture the linear effects of unobserved project heterogeneity at the cell level. Our primary interest, however, is in the *interaction coefficients* between the developer type and the receiving project characteristics, the $\lambda's$. These coefficients describe how developers endogenously sort (i.e., target their contributions) on the license type and other receiving project characteristics. We refer to these interaction coefficients as the "sorting parameters."

We use the following dimensions to define cells. The first dimension is developer type. As explained earlier, we infer the developer type from the developer's project affiliations (membership). Using the contributing developer type allows us to examine whether developers sort—that is, target—projects with specific types of open source licenses. For the receiving project, we use five characteristics: license type, intended audience, programming language, operating system, and age. Project age is important because we measure the number of contributions made over the entire life of the project and this will depend on how long the project has been registered. Cell dimension are defined as dummy variables. An example of a cell is the total number of contributions open developers make to projects with a closed license and a particular intended audience, operating system, programming language, and project age. The total number of cells in the regression equals the product of the number of developer types, intended audiences, operating systems, programming languages, and project ages for which information on contributions is available. Some cells have only dormant projects which receive no contributions over the sample period, in which case we drop them in the estimation procedure.

As explanatory variables in the regressions, we include a complete set of dummy variables for the receiving project characteristics, and the interactions of the receiving project license type and intended audience with developer type. In addition, we include the average size (number of registered members) of the receiving projects in each cell, interacted with the developer type, to allow for sorting by developers on project size. We also control for the number of potentially receiving projects in the cell since that will affect the flow of contributions. We do not control for the number of potentially contributing developers because there is no way to do this for Anonymous developers. This means that it is difficult to interpret differences in the levels of sorting coefficients *across* developer types. Our main focus will be on how each developer type sorts across the various business model dimensions of the project, including license type, size, and intended audience.

We must normalize one of the coefficients on the interaction dummy variables between developer and license type (as we also control for additive effects). The choice of normalization only affects the interpretation, not the estimation, of parameters. We set the coefficient on the open developer–HO license interaction equal to zero, so all estimated interaction coefficients measure impacts relative to this reference group, that is, relative to the expected number of contributions by open developers to projects with HO licenses. Each sorting coefficient gives the impact of a unit change in the control variable on the expected number of contributions, all defined relative to the number of contributions contributed by the open developers to projects with HO licenses (the reference category). Sorting behavior by a developer type is revealed by comparing the sorting coefficients for that developer type across projects with different licenses and other business model dimensions.

Our identification assumption is that the license types, and other project characteristics, are exogenous with respect to the *individual developer's* decision to contribute. The main concern is unobserved project quality, which might be correlated with both the observed characteristics of, and the number of contributions made to, a project. Our empirical strategy should be more robust to this problem, however, because our primary focus is on the *interactions* between the contributing developer type and project characteristics, the λ coefficients. Unobserved heterogeneity will induce bias only if it is correlated with these interactions. While higher-quality projects may attract more contributions, why this should systematically affect one type of contributor more than the other is unclear. Nonetheless, in the robustness checks section we discuss a battery of tests we performed to check the sensitivity of our results for developer and project unobserved heterogeneity.

7. Descriptive Statistics

Developers in our sample make 103,712 external source code contributions. Table 1 shows how these contributions distribute across developer types. Of the total, 31% come from non-members, 18% from anonymous contributors, 14% from open developers, and 12% from closed developers. The remaining contributions are from mixed developers. Mixed developers are the most active contributors, with an average of 11.2 contributions per developer. The least active developers are non-members, with only 2.1 contributions per developer.

Table 2 summarizes key aspects of sorting behavior by developers. We find strong sorting of contributions on project license type. Both anonymous and open developers are much more likely to contribute to projects with highly open licenses. More than 80% of anonymous developers' contributions go to such projects, with a further 10% or more directed to moderately open projects. Open developers follow a similar pattern: two thirds of their contributions go to highly open projects, with an additional 17% going to moderately open projects. In sharp contrast, 36% of closed developers' contributions go to projects with closed licenses (as compared to 9% and 16% for anonymous and open developers, respectively).

We also find strong sorting on the intended audience. Closed developers are 3.5 times more likely to contribute to projects aimed at developing programming tools than to projects whose main audience is end users. Anonymous developers, meanwhile, are almost four times more likely to contribute to projects aimed at end users than projects targeting other developers. Open developers do not seem to favor either end-user or developer projects. Lastly, closed developers sort more strongly on the size of the project compared to anonymous or open developers. 43.9% of contributions by closed developers go to projects with more than 10 members, the corresponding figures are 30.7% for open developers and only 17.7% for anonymous contributors.

8. Econometric Results

8.1. Sorting by Project License Type and Intended Audience

Table 3 presents the estimated marginal effects for the baseline model. We focus on the sorting coefficients that describe matching between the contributing developer type and the license and size of the receiving project. We find a strong sorting behavior by license type. Turning first to column 2, open developers are much more likely to contribute to projects that have highly open licenses than to those with less open licenses. Changing the project license from highly open to open reduces the number of contributions by open developers by 1.62, or 17.4%. Moving from an open to a closed license is associated with an additional reduction in contributions by such developers of 0.34, or 3.7%. The χ^2 test strongly rejects the hypothesis that no sorting occurs by license type for highly open developers (p-value$< .001$).

Career concerns cannot drive contributions by anonymous developers, because these developers' anonymity prevents gain from peer recognition or labor market signaling. Thus anonymous contributions indicate either the importance of intrinsic motivation or pure utility/learning value from contributing. If they are primarily intrinsically motivated agents, we expect them to sort on highly open projects, whereas the utility value/learning incentive predicts no systematic sorting on license type. Column 1 shows that anonymous developers sort in a way similar to highly open developers; both show a similar sorting coefficient on HO licenses and a strong disinclination to contribute to projects with less open or closed licenses. The statistically significant, negative sorting on open and closed licenses (-2.20 and -2.95, respectively) shows this sorting. These results for highly open and anonymous developers provide support for the "pure intrinsic motivation" hypothesis, indicating that these types of developers attach value to the open source ideology that favors highly restrictive (open) licenses. This evidence strongly supports Hypothesis 1a.

We find the opposite pattern of sorting by closed developers (column 4), who are much more likely to contribute to projects with less open or closed licenses. For example, comparing the sorting coefficients for the C and HO licenses (-1.40 and -2.93, respectively), we see that moving from a C to an HO license reduces the number of contributions by closed developers by 1.53. This represents an 18.7% fall in their contributions at the cell level (= 1.53/8.2). Again, we decisively reject the null hypothesis that no sorting by closed developers (p-value <.001) occurred. This evidence is consistent with Hypothesis 1b.

Interestingly, we do not find any sorting behavior for mixed developers (column 3), who are registered members of projects with different types of licenses. We cannot reject the null hypothesis that there is no sorting for these developers (p-value = .51). This indicates an indifference to the choice of project license type, both in their choices regarding membership and in their contribution activity.

Column 5 shows that non-member contributors exhibit sorting behavior broadly similar to (but less sharp) open and anonymous developers. In particular, non-members sort toward highly open licenses. Moving from an HO to a C license reduces by 11.6% the number of contributions by non-members.

We turn next to the impact of project size on the inflow of contributions. Project size plays two roles. First, the number of members who belong to a project may be a proxy for unobserved project quality, in which case, larger projects would attract more contributions from all developer types. Second, project size may be associated with more exposure and thus larger reputation gains. But these gains can come in the form of greater peer recognition and/or in labor market signaling benefits, and thus can be enjoyed both by open developers and more commercially oriented, closed developers. Hence there is no theoretical prediction as to whether open or closed developers should value project size more strongly. This is an empirical question.

We can distinguish between the project quality and reputation effects associated with project size by exploiting the anonymous developers. Reputation benefits should not be relevant to anonymous developers, but they may prefer higher-quality projects. Therefore, we can infer the impact of project quality on contributions from the behavior of the anonymous type. Assuming that other developer types have similar preferences for contributing to high-quality projects, we can identify the reputation effect associated with project size by taking the coefficient on size for each developer type minus the corresponding coefficient for anonymous developers. We find a statistically significant effect of project size for anonymous developers, but the impact is not large. The point estimate of 2.66 implies that a 10% increase in project size raises contributions by 0.27, which is only 2.2% of the average number of contributions at the cell level by these developers. The fact that size matters at all for them, however, is interesting because it indicates that the utility gains from contributing are related to project size (or project quality, for which it may serve as a proxy). Subtracting the point estimate for the anonymous developers from the size coefficients for the other developer types, we get the following estimates for how reputation gains are related to receiving project size: 0.03 for open developers, 0.31 for mixed, 0.91 for closed, and 0.56 for non-members. These coefficients indicate that reputation gains for open developers do not appear to be related to project size, as we measure it, but they are related for the other developer types, especially the more commercially oriented, closed developers. This finding strongly supports Hypotheses 2a on reputation and project size.

To test Hypothesis 2d, on reputation and intended audience, Table 4 explores sorting patterns on intended audience.[9] We add interaction terms for each developer type with their intended audience dummies: developer tools and end users. The results show strong sorting on intended audience by both open and closed developers. Starting with open developers, shown in column 2, moving from developer tools to end-user projects is associated with an increase of 2.12 (= 0.65+1.47) in the number of contributions. This change in intended audience type reduces their contributions by 22.8% (relative to the cell average number of contributions). Anonymous developers show a similar pattern. Moving from developer tools to end users raises their number of contributions by 1.25 (= -1.56+2.81), or 10.3% of the average cell number of contributions. By contrast, closed developers display a strong preference for developer-tool projects. Moving from end users to developer tools increases the number of contributions made by closed developers by 1.91 (= 2.05-0.14), accounting for 23.3% of contributions by this developer type. Interestingly, non-member developers continue to be similar to open developers in terms of intended audience sorting, while the sorting pattern for mixed developers is similar to that of closed developers.

Sorting by intended audience can be driven by other factors that are not related to reputational concerns. We can control for their effects using the sorting results for anonymous developers, similarly to what we did with project size. Because reputation is not relevant as a motivation for anonymous developers, the sorting coefficient for this category picks up all the non-reputation effects that are related to intended audience. Thus, we can isolate the reputation effect associated with sorting by intended audience by subtracting the sorting coefficient of anonymous for each developer type. This calculations yields a reputation effect of -0.87 (= -2.12+1.25) for open developers, and 3.16 for closed developers (=1.91+1.25). Standard errors on these differenced coefficients indicate a statistically insignificant reputation effects for open developers, but a statistically significant reputation effects (at the 5% level) for closed developers.

These findings provide partial support for Hypothesis 2d. We find the predicted sorting by closed devel-

[9]Note that license type is highly correlated with intended audience, which makes identifying sorting by license type and intended audience more difficult. In our sample, 87% of end-user projects have a highly open license, as compared to only 31% for developer projects.

opers toward developer-tool projects, where labor market signaling benefits are likely to be larger. However, we also find sorting by open and anonymous developers toward end-user projects, whereas we predicted that open developers would favor developer tools and anonymous would not sort on this dimension.

8.2. Quantifying the Effect of Project License Type on Sorting and Project Performance

In this section, we demonstrate how information about individual motivations can be strategically exploited by firms in choosing the openness of their project licenses. For this purpose, we estimate a project performance specification to obtain estimates of the marginal productivity of different developer types. As our measure of project performance, we use the number of times the project has been downloaded. Project dissemination in open source is likely to be a main objective of project managers, and downloads are one strong indicator of project quality.

We estimate a log-linear function that relates the number of downloads of a project i in year t, Y_{it}, to the aggregate stock of contributions it receives, S_{it}, and a set of other control variables, which we denote by Z. The specification can be expressed as

$$\ln Y_{it} = \alpha \ln S_{it} + \beta Z_{it} + \eta_{it}$$

where η is a normally distributed error term that we assume to be independent of $\ln S$ and Z. In specifying the appropriate aggregate stock of contributions, however, we do not simply add up all past contributions regardless of their type. Instead, we treat different types of contributions as perfect substitutes but allow their marginal productivities to differ. Specifically, we use $S_{it} = \Sigma_{j=1}^{J} \lambda_j S_{ijt}$, where S_{ijt} is the stock of contributions of type j for project i in year t.

Substituting this expression for the stock of contributions, we get the following estimating equation for performance:

$$\ln Y_{it} = \alpha \ln (\Sigma_{j=1}^{J} \lambda_j S_{ijt}) + \beta Z_{it} + \eta_{it} \tag{8.1}$$

We need to normalize one of the λ parameters—we set $\lambda_{Non-members} = 1$,which means that the parameter λ_j represents the marginal productivity of the stock of contributions of type j *relative to* the marginal productivity of contributions of non-members. The coefficient α is the elasticity of downloads with respect to the aggregate stock of contributions. Each stock is computed as the cumulative number of contributions of the specified type from the project's inception. We include controls for intended audience, programming language, operating system, and year of registration. We estimate this regression by nonlinear least squares and report standard errors clustered at the project level.

Table 5 presents the parameter estimates. In column 1, we begin with a simple specification using only the total stock of contributions received by the project, regardless of source. We find that project performance is strongly related to contributions received, with a statistically significant elasticity of 0.307. This estimate means a 10% increase in the stock of contributions raises the number of downloads per year by 3.1%. This underlines the importance of choices project managers make that influence the flow of contributions.

The association between the stock of contributions and downloads hides an important variation. In column 2, we break contributions into three separate stocks: *internal contributions* (from developers who are members of the same project), *external contributions* (from developers who do not belong to the project), and *non-member contributions.* The result is striking—whereas the overall elasticity is virtually unchanged, at 0.31, the estimated marginal productivities of both internal and external contributions are much smaller than for non-member contributions (which is normalized to unity). The estimated relative marginal productivity for internal contributions is 0.34, only a third as large as for non-member developers, whereas the relative marginal productivity for external contributions is 0.62. We cannot reject the hypothesis that internal and external contributions have the same marginal productivity (p-value =0.08), but we strongly reject that they are the same as for non-member developers.

Column 3 breaks the stock of external contributions into three types: open and anonymous, closed, and mixed, and shows that the marginal productivities of these contributors differ sharply. The striking finding is that the marginal productivity of open and anonymous developers is very similar to the reference category, non-members, as evident by the coefficient estimate of 1.10. By sharp contrast, closed and mixed developers look very similar to each other but their marginal productivity is far lower than the other categories of developers.

We now use these estimated parameters to compute the actual (not relative) marginal productivity for each type.[10] For non-members, our normalized category, the marginal productivity is 2.33, which means an additional contribution from non-member developers is associated with 2.33 more project downloads. The marginal productivity for anonymous and open developers is 2.57. For closed and mixed developers, the marginal productivity is much lower at 0.25 and 0.26, respectively. We will now use these estimated marginal productivities to compute the effect of sorting on downloads.

This analysis reveals two key facts. First, contributions matter for project performance. Second, contributions by intrinsically motivated (open) developers – for which the tradeoff between forgone property

[10]The marginal productivity of developer type j is computed as $MP_j = \alpha\lambda_j \frac{Y_{it}}{\Sigma_{j=1}^{J}\lambda_j S_{ijt}}$. We evaluate all marginal effects at the sample average values.

rights and the flow of contributions is important – have the highest marginal productivity. This result is important for managerial strategy because it underscores the benefits associated with designing business models to tap into this highly productive part of the open source labor force. Managers must decide whether these benefits balance the costs of such sorting, which are the costs of forgone property rights a more open license imposes.[11]

We turn next to compute the effect of project openness on sorting and project performance. Let $\Delta^c_{l \longrightarrow L}$ denote the change in the number of contributions received that occurs when license type shifts from l to L, holding all other characteristics of the business model constant. Let λ_{dl} denote the sorting coefficient for developer type d and receiving project (cell) l. This parameter is the expected number of contributions by all developers of type d to projects with license l, relative to the normalized group, which is contributions by open developers to projects with an HO license ($\lambda_{O,HO} = 0$). Then the effect of changing the license from type l to L on the total expected number of contributions is given by $\Delta^c_{l \rightarrow L} = \sum_d (\lambda_{dL} - \lambda_{dl})$, where the summation is over the five different developer groups. The respective effect of this change on project performance is $\Delta^p_{l \rightarrow L} = \sum_d MP_d(\lambda_{dL} - \lambda_{dl})$, where MP_d denotes the marginal productivity of developer d (which is reported above).

Using these formulations and our estimates from Tables 3 and 5, we find that moving from a HO to C license leads to a loss of 15 contributions and 39 downloads per project (30% of median number of downloads). Moving from a HO to O license leads to a loss of 7.8 contributions and 20.6 downloads (16% of median number downloads). Moving from O to C projects reduces the number of contributions by 7.8 and number of downloads by 17.4 (13% of median number of downloads). These calculations indicate the HO licenses maximize the expected number of contributions and project downloads. Of course, this result does not mean that managers should always choose HO licenses, because such licenses also involve constraints on the ability to appropriate commercial returns from the project. But the computation shows that the type of project license can make a real difference to the flow and impact of contributions, and thus the rate of innovation in OSS.[12]

[11] An interesting finding is the large marginal productivity for non-members (larger than any other developer type except open developers), and the fact that they make the smallest number of contributions (Table 1, Column 4). A potential explanation of this pattern is that non-member developers are generalists problem solvers –the kind of knowledge workers that do not commit themselves to a specific project by becoming formal members, but rather identify and solve high-quality problems (as indicated by their impact on performance) in several different projects. Our sorting analysis shows that these "generalist problem solvers" sort on license type and tend to prefer open projects. Our results imply that managers who wish to tap into these productive, generalist problem solvers in open innovation systems need to use open licenses, which of course is costly in terms of forgone property rights.

[12] Our results also have implications for the joint selection of different dimensions of the business model. We showed that choosing a highly open license increases the share of contributions from intrinsically motivated developers. We also showed, in Table 4, that intrinsically motivated developers sort on projects whose intended audience is end users. Therefore, to attract

8.3. Extensions

In this section, we discuss two additional business model choices to refine our inferences about the role of reputation.

8.3.1. Public resolution of contributions

Reputation building is likely to be an important driver of contributions (our previous findings with respect to the interaction between developer type and project size point in this direction). Our first extension demonstrates how the project manager can rather easily influence reputation concerns associated with contributing to her project. We exploit information on whether the receiving project announces whether a contribution is accepted. After a developer makes a contribution, the project manager decides whether or not to accept it and to make the decision public on SourceForge. The manager makes the decision separately for each contribution, but projects differ in the degree to which they make these outcomes public.[13] Thus, whether to reveal the outcome of a contribution is a business model feature, which we show has strong consequences for sorting.

We explore two theoretical predictions, both embodied in Hypothesis 2b. The first is that anonymous developers' decision to contribute should not be affected by whether projects publish the decision, because they cannot benefit from any peer-based or commercial reputation gains. However, if anonymous developers are also driven by intrinsic motivation (including open source ideology), publishing the outcome might be viewed as an extrinsic motivation device that actually crowds out their intrinsic desire to contribute to such projects. In this case, contributions by anonymous developers should be lower for projects with public resolution.

Developers motivated by reputation should be more likely than other developers to contribute to projects with public resolution. Thus we can test directly whether reputation matters by examining whether contributions by non-anonymous developers are higher for such projects. However, because both peer-based and commercial (labor market) reputations can be at work, we have no *a priori* prediction about *which* types of developers should be most sensitive to public resolution.

To study these hypotheses, we define a dummy variable equal to one for projects that reveal the outcome

these developers, we would expect managers of end user projects to be more likely to adopt HO licenses, as compared to other intended audiences. Our data confirm this prediction. For end user projects, HO licenses are dominant: 87% have highly open licenses, with O and C licenses accounting for only 5% and 8%, respectively. However, for developer tool projects, there is a more even distribution across license types: 31% for highly open, 33% for open, and 36% for closed licenses.

[13] In total, these projects receive 68,294 "closed" contributions, of which 14,147 (20.7%) have no reported resolution, 45,844 (67.1%) have an "Accepted" resolution, and the remaining 8,303 contributions get a "Rejected" resolution. Overall, 67.7% of projects receiving contributions publish the resolution to some degree.

for at least 50% of the contributions they receive over the sample period.[14] With this threshold, we classify 71.1% of the projects as having public resolution. Projects that disclose resolution tend to be larger than projects that do not (mean project sizes are 5.7 and 4.2, respectively; mean number of contributions received are 16.1 and 7.8). We add this public resolution dummy as an additional dimension for defining the cells for the estimation procedure, and re-estimate the baseline specification.

Table 6 presents the results. Our earlier findings about sorting on license type are similar to the baseline results in Table 3. Two new findings arise here. First, the estimated coefficient on the public resolution dummy variable is positive and statistically significant for all groups *except* anonymous developers. This result confirms the hypothesis that reputation is a motivation for contributions, and confirms Hypothesis 2b. Public resolution has the largest impact for open and closed developers. The estimated coefficient of 3.09 for open developers implies they contribute 60% more on average to projects with public resolution (= 4.01/6.7), while for closed developers the figure is 54% (= 3.15/5.8).

The second important finding is that the estimated coefficient for anonymous developers is negative and significant. These developers do not value public resolution, which is consistent with the theoretical prediction, because they do not enjoy the reputation gains. More striking is the fact that anonymous developers are actually less likely to contribute to projects with public resolution, which indicates such publication may crowd out the intrinsic motivation underlying anonymous contributions. This finding is robust to using alternative thresholds for classifying projects as having public resolution.

8.3.2. Corporate sponsorship

Increasingly, large firms have invested substantial resources in open source development, including paying employees to participate in such projects (Lerner and Schankerman, 2010; Fosfuri et al., 2008), and is also likely to include sponsorship and other forms of involvement with projects registered on SourceForge. Knowing which projects have substantial corporate involvement should help us pin down more sharply the role of labor market signaling, as distinct from peer recognition. Moreover, corporate sponsorship is an important decision of the project manager. Although it is likely to be associated with significant monetary benefits, it is also likely to have implications for the types of developers that would choose to participate, either as members or external contributors, in the project development. We explore this implication in this section.

The main prediction (Hypothesis 2c) is that developers motivated by commercial reputation—in particu-

[14] We also tried two alternative thresholds—25% and 75%—to define the dummy variable for public resolution. The main conclusions from this analysis are robust to the choice of the threshold, though the parameter estimates differ somewhat.

lar, closed developers—should be more likely to contribute to projects with corporate sponsorship, conditional on the license type of the receiving project. Corporate sponsorship should have a zero effect on anonymous developers, because labor market signaling plays no role for them, or a negative effect if sponsorship is viewed as an extrinsic payoff and crowds out of intrinsic motivation. Similarly, unless labor market reputation matters for open developers, we expect corporate sponsorship to either have a zero effect on their contributions or a negative effect if they are "motivated agents" with anti-proprietary software ideology.

SourceForge does not identify whether a project is corporate sponsored, or more generally the level of corporate involvement. We sent an e-mail survey to registered members of the largest 1,000 projects (measured by number of contributions received) listed on SourceForge to determine whether projects were initiated by for-profit companies or not-for-profit organizations. We received responses for 217 projects, but the information only allowed us to identify the status clearly for 93 projects. To augment the usable sample, we performed extensive manual investigation of each remaining project to identify whether corporate involvement was present, either directly or through offering a proprietary product that incorporates source code from the project. Because both forms of engagement provide opportunities to developers for labor market signaling, we classify such projects as having a "corporate sponsor." We also checked whether projects had a clear not-for-profit mission. This investigation allowed us to increase the sample to 148 projects (there are 45,687 contribution received by these projects).[15]

These projects are typically the larger and more active projects on SourceForge—the median number of members is 22, and these 148 projects account for about 45% of all contributions in the complete sample. The distribution of contributions by developer types and corporate sponsorship (Table 7) shows clear sorting behavior. Anonymous developers target 86% of their contributions to not-for-profit projects, while 79% of contributions by closed developers go to corporate-sponsored projects. Open developers favor not-for-profit projects (55%), but this sorting is weaker. This pattern indicates some role for signaling and other labor market considerations even for open developers.[16]

To analyze sorting on corporate sponsorship more formally, we estimate a Probit model that relates whether a contribution goes to a corporate project (dependent variable equal to one) or a not-for-profit project, conditional on dummy variables for developer type and receiving project characteristics.[17] Table 8

[15]Importantly, not-for-profit was not our default option. To be so classified, a project had to indicate clearly a non-commercial intent in its mission on its website or the various online forums we examined, and no company support (or stated intention to seek it) could exist, nor any commercial products we could identify as building on the project.

[16]One example of a project to which both open and closed developers contribute is Jboss. This project is incorporated in RedHat's products, such as Jboss Entreprise Middleware. Interestingly, anonymous contributors are the only type that almost never contribute to this project, confirming their strong intrinsic motivation in favor of highly open (and non-corporate) projects.

[17]We do not adopt the "cell" framework in this section because, despite the large number of contributions covered by the

summarizes the results and confirms the pattern found in Table 7. In column, 1 we include only dummies for developer types. Closed developers are 32.4 percentage points (or 65.7%, evaluated at the sample mean) more likely than open developers to contribute to corporate projects. Anonymous developers are 36.2 percentage points (or 73.4%, evaluated at the sample mean) less likely than open developers to engage with corporate projects. In column 2, we add a control for project size and find the same pattern of results. In column 3, we add dummies to control for the license type of the receiving project and find no significant change. Interestingly, the estimates also show that corporate projects are least likely to use a highly open license. In column 4, we add dummy controls for the intended audience, programming language, and operating system of the receiving project. Even with these controls, we find the same sorting of closed developers toward corporate projects, relative to open developers, but the magnitude is smaller by about half. In addition, with these controls, the difference between anonymous and open developers vanishes. These results strongly confirm Hypothesis 2c.

8.4. Evidence on Reciprocity

In this section we analyze the role of reciprocity in open source innovation. Early proponents of open source software argued that reciprocity was an important motive for developers to contribute, and one that would be self-sustaining (e.g., Raymond, 2001). Similar claims have been made in other, so-called "gift-culture," settings. However, to our knowledge, this is the first empirical evidence of reciprocity in the software context.

We focus on reciprocity at the project level and define a contribution to project j in year t from a developer who is a member of project i as reciprocal if project i received a contribution from j prior to year t. We measure the degree of reciprocity as the percentage of all contributions made by project i that go to projects that previously contributed to it.

Table 9 presents descriptive statistics on reciprocity. Three features stand out. First, reciprocity is rare at the project level. For the complete sample, only 4.9% of active projects (those that receive at least one contribution) are engaged in reciprocity (Column 1). However, these tend to be larger projects, accounting for 37% of total contributions received (Column 2). Second, closed projects are more likely to involve reciprocity—7.5% for closed versus 4% for highly open projects (Column 2). This is also true in terms of the percentage of contributions—closed projects with reciprocity account for 74.8% of received contributions, compared to 20.6% for highly open projects (Column 3). Finally, whereas only a small minority of projects

projects in this restricted sample, the degrees of freedom used for cell construction dependent almost solely on the number of receiving projects, regardless of how many contributions they receive. In the current sample, we have only 148 (mostly large) projects, which is too few to meaningfully break up by bundles of project characteristics.

engage in reciprocity, it plays a large role for those that do. Among those projects, 44.6% of all contributions made are reciprocal; this is again most pronounced for closed projects (Column 4).

We proceed by estimating Probit regressions of whether a contribution is reciprocal. To control for project similarity, we include a set of dummy variables that capture whether the projects *match* on license type, programming language, operating system, and intended audience. We also control for additive dummies for each of these dimensions. Table 10 presents the results, which confirm the non-parametric evidence from Table 9. Column 1 shows that reciprocity is more likely when there is matching on license type and intended audience. The impacts are substantial, especially for license type—matching on license type raises the likelihood of reciprocity by 12 percentage points, or 64.2% of its sample mean. However, matching on programming language or operating system does not affect reciprocity. This suggests that reciprocity is not primarily driven by project similarities that reduce the cost of making contributions. These results continue to hold when we include the size of the projects (column 2). In column 3, we look more closely at matching on license type, including separate dummy variables for matching on different types of licenses. What is striking is that reciprocity is much higher when there is matching on closed licenses, but we find no effect when projects are matched on more open license types.

These results are not consistent with Hypothesis 3a, because reciprocity is not the prevalent mechanism driving contributions, though it is important for some projects. Further, our findings that reciprocity is more important for closed projects confirms Hypothesis 3c, but not 3b. From a business standpoint, our reciprocity analysis shows that when managers choose a closed project license, they help create conditions that support a reciprocal relationship formation with other developers and projects, which shapes the pattern of future source code exchanges.

8.5. Robustness Analysis

This section summarizes robustness analysis we performed to address concerns about potential endogeneity issues. The detailed set of results from this analysis is provided in the online appendix. A short summary of the approaches we take to address endogeneity is presented in Table 11.

Developer experience

Our main specifications do not control for within-cell developer heterogeneity. To check the sensitivity of our results, we control for developer experience. Experience is defined as the stock of contributions made by a developer up to the date of the focal contribution. We find that sorting behavior is actually stronger for more experienced developers (Table A2 in the online appendix). Among the most experienced developers (top quartile), open developers direct 95% of their contributions to highly open projects, and closed developers direct 86% of their contributions to closed projects. For the lowest quartile, these figures are 73% and 45%. Finding that sorting increases with the level of experience is an important result because it shows that the labor market implications of property rights choice are especially strong for the more experienced (and presumably higher quality) developers.

To examine further the robustness of our results to developer experience, we add an experience control in our baseline estimation (Table A3a in the online appendix). In this analysis, we are forced to exclude the anonymous category and calculate for each cell the average level of developer experience. We find that experience has a strong positive effect on the propensity to contribute, as expected. Importantly, sorting by license type remains robust.

Lastly, because we find that the least experienced developers have the weakest sorting pattern, we are concerned that in the pooled regression the sorting parameters may reflect differences between the highly experienced and the least experienced developers. To mitigate this concern, we estimate our baseline specification for a sample that excludes the least experienced developers. The sorting parameter estimates remain robust (Table A3b in the online appendix).

Matching on programming skills and software architecture

An important concern is that the observed sorting behavior might be influenced by similarity in programming language or software architecture, which developers seek to exploit but which might be correlated with project license type. If developers are matching their skills to the programming language or software architecture (e.g., operating system), we might be confounding the mechanism that induces sorting. To address this concern, we estimate our baseline specification (from Table 3) at the contributing and receiving project level. In this specification, we define cells on the basis of the characteristics of the project from which the contribution is coming and the project to which it is directed (this increases the number of cells to 374,808). The contributing project is the project with which the developer making the source code contribution is affiliated as a registered member. Cells now include detailed information on additional dimensions that can drive contribution, such as contributing and receiving programming language, operating systems and intended audience (in the baseline specification, the only control we included for contributing characteristics was developer type). We include complete sets of dummies for all of these contributing and receiving characteristics. In this analysis, we cannot include contributions made by anonymous and non-member developers, which is one of the main reasons we do not use this specification as our main analysis. Column 1 in Table A4 (online appendix) shows that strong sorting on license type also holds when controlling for linear effects of receiving and contributing programming language, intended audience and operating systems.

To address the concern that matching by programming language drives sorting, Column 2 in Table A4 includes a dummy that equals 1 for cells containing a matching on programming language; that is, the programming language of the contributing project is the same as the programming language of the receiving project. The coefficient estimate on this dummy is, not surprisingly, positive and highly significant. Our findings on matching on license type remain robust. Column 3 allows matching to vary by specific programming languages by adding five dummy variables for matching by programming language separately for each language category. Sorting by licensing type remains robust.

Our next test constructs a measure of the proximity between each pair of programming languages, in the spirit of Jaffe (1986). Let $T_l = (t_{1,...,}t_5)$, where t_i is the share of contributions that projects using programming language l receive from projects that use language i. The proximity between languages l and l' is the (uncentered) correlation coefficient between T_l and $T_{l'}$. This measure varies between zero and one with higher values representing closer proximity. We include this measure as a control in the regression. The estimated marginal effect of this measure is positive and highly significant as shown in Column 3. However, our results of sorting by license type remain robust.[18]

Lastly, Column 5 presents the results of the most flexible specification we can estimate in this context, where we include a complete set of dummy variables for all possible combinations between pairs of contributing and receiving programming languages. The license type sorting parameters remain robust in this specification as well.

Project quality

We have no obvious reason to expect that unobserved project quality is systematically correlated with the interaction between project license and developer type. Nonetheless, we check robustness by introducing two direct measures of project quality. The first is the cumulative number of project downloads *prior* to the year of contribution. The second is the lag between contributing year and project registration year. The idea behind this second measure is that higher quality projects are more likely to attract contributions for a longer period of time (rather than just when the project is launched). In the baseline analysis, we did not use information on the year of contributions in the definition of cells. However, to conduct these additional tests, we need to redefine cells more finely, including the year of contribution as an additional dimension.

[18] To check further the robustness of our results, we construct an additional Jaffe-type measure, which is based on the distribution of contributing developers. The idea behind this measure is that two programming languages are likely to be "closer" if the distributions of contributions received from individual developers are more similar. The length of the new vector is n, where n is the total number of contributing developers in our sample. Including this developer-level proximity measure does not affect the sorting parameters

The main results on sorting hold when we introduce these controls for project quality (results are omitted for brevity). We find that past cumulative downloads are positively, and significantly, related to the number of contributions received by a project in a given year—consistent with downloads being an indicator of project quality—but we find no statistically significant effect for the contributions lag. Importantly, the pattern and magnitude of sorting are robust to these controls.

Variation over time and removing outliers. Corporate involvement in the open source community has been increasing, which may be a reflection of a less sharp ideological divide between the open source and proprietary software communities. Therefore, we check the sensitivity of the sorting effects to temporal shifts. We estimate the baseline model separately for the periods 1999–2005 and 2006–2010, based on project registration year. We find that sorting on license type holds for both periods, but sorting is actually stronger in the 2006–2010 period.

Lastly, the distribution of contributions is highly skewed—a few projects receive a large fraction of contributions, and a few developers make a significant share of contributions. Because unobserved heterogeneity may be present in these groups, we check the robustness of the results to removing outlier projects and developers. We drop projects that receive a very large number of contributions and developers who make a very high number of contributions (in each case, we winsorize at the 99th percentile). In both cases, the same pattern of sorting continues to hold.[19]

9. Concluding Remarks

This paper demonstrates that business model design has important labor market implications, especially when intrinsically motivated workers are significant elements in the labor force. We show that there is an important tradeoff between stronger property rights associated with certain types of licenses, which managers choose, and the flow of contributions the project receives from the most productive, intrinsically motivated, developers. Using a large dataset on the pattern of contributions in OSS, we find that different types of developers sort strongly on observed project characteristics—most notably, openness of the license, intended audience, project size, and corporate sponsorship. The pattern of sorting behavior points to an important role for intrinsically motivated agents, as well as reputation, especially commercial reputation for closed developers. We find an important role for reciprocity, but only for a small set of projects and commercially oriented developers. We show how managers can strategically exploit this sorting behavior to improve the

[19]To check the sensitivity of our results to empty cells (zero values), we re-estimate our baseline model using a Zero-Inflated Negative Binomial model (ZINB). The sorting parameters remain robust. Importantly, we do not reject the Vuong test for whether ZINB is preferred over Negative Binomial (p-value=0.97).

performance of their projects.

There are two main directions for future research. The first challenge is to understand better why firms choose different business models in OSS, in light of our finding that open licenses have the benefit of attracting the highly productive, intrinsically motivated developers, but at the cost of weaker property rights. The central question is why this tradeoff operates differently for different firms. The second important issue is to understand the economic and behavioral micro foundations for our finding that reciprocity appears to be more common among commercially oriented projects, and to explore what this implies about the need for formal property rights to facilitate technology exchange in open innovation systems.

References

[1] Arora, A., Fosfuri, A., and Gambardella, A. 2001. *Markets for technology: The economics of innovation and corporate strategy.* Cambridge, MA: MIT Press.

[2] Baldwin, C. and von Hippel, E. 2011. Modeling a paradigm shift: from producer innovation to user and open collaborative innovation. *Organization Science*, 22(6): 1399-1417.

[3] Bénabou, R., and Tirole, J. 2003. Intrinsic and extrinsic motivation. *Review of Economic Studies*, 70(3): 489–520.

[4] Besley, T., and Ghatak, M. 2005. Competition and incentives with motivated agents. *American Economic Review*, 95(3): 616–36.

[5] Boudreau, K., Lacetera, N., and Lakhani, R. 2011. Incentives and problem uncertainty in innovation contests: An empirical analysis. *Management Science*, 57(5): 843–863.

[6] Chesbrough, H. 2003. *Open innovation: The new imperative for creating and profiting from technology.* Cambridge, MA: Harvard Business Press.

[7] Coff, R. W. 1997. Human assets and management dilemmas: Coping with hazards on the road to resource-based theory. *Academy of Management Review*, 374-402.

[8] Coff, R. and Kryscynski, D. 2011. Invited editorial: drilling for micro-foundations of human capital–based competitive advantages. *Journal of Management*, 37(5), 1429-1443.

[9] Cohen, W. 2005. Does Open Source Have Legs? in *Intellectual Property Rights in Frontier Industries*, Robert Hahn, ed., Brookings.

[10] Deci, E. 1972. Intrinsic motivation, extrinsic reinforcement, and inequality. *Journal of Personality and Social Psychology*, 22: 113–120.

[11] Fosfuri, A., Giarratana, M. and Luzzi, A. 2008. The Penguin Has Entered the Building: The Commercialization of Open Source Software Products. *Organization Science*, 19(2): 292–305

[12] Gambardella, A., Panico, C. and Panico, G. Strategic Incentives to Human Capital. *Strategic Management Journal*, Forthcoming.

[13] Gottschalg, O., & Zollo, M. 2007. Interest alignment and competitive advantage. *Academy of Management Review*, 32(2), 418-437.

[14] Giuri, P., Ploner, M, Rullani, F. and Torrisi, S. 2010. Skills, division of labor and performance in collective inventions: Evidence from open source software. *International Journal of Industrial Organization*, 28(1): 54-68.

[15] Haruvy, E., Wu, F., and Chakravarty, S. 2003. Incentives for developers' contributions and product performance metrics in open source development: An empirical investigation. Working paper, University of Texas at Dallas.

[16] Hertel, G., Krishnan, M., and Slaughter, S. 2003. Motivation in open source projects: An Internet-based survey of contributors to the Linux Kernel. *Research Policy*, 32(7): 1159–77.

[17] Jeppesen, L., and Lakhani, K. 2010. Marginality and problem-solving effectiveness in broadcast search. *Organization Science* 21: 1016–1033.

[18] Johnson, J. 2002. Open source software: Private provision of a public good. *Journal of Economics and Management Strategy*, 11: 637–62.

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

[20] Lakhani, K. and von Hippel, E. 2003. How open source software works: "free" user-to-user assistance. *Research Policy*, 32: 923–43.

[21] Lerner, J. and Schankerman, M. 2010. *The comingled code: Open source and economic development.* Cambridge: MIT Press.

[22] Lerner, J. and Tirole, J. 2001. The open source movement: Key research questions. *European Economic Review*, 45(4–6): 819–26.

[23] Lerner, J. and Tirole, J. 2002. Some simple economics of open source. *Journal of Industrial Economics*, 52: 197–234.

[24] Lerner, J., Pathak, P. and Tirole, J. 2006. The Dynamics of Open-Source Contributors. *American Economic Review*, 96(2): 114-118.

[25] Raymond, E. 2001. *The Cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary.* Cambridge: O'Reilly Press.

[26] Roberts, J., Hann, I-H., and Slaughter, S. 2006. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects. *Management Science*, 52(7): 984–999.

[27] Sauermann, H. and Cohen, W. M. 2010. What makes them tick? Employee motives and firm innovation. *Management Science*, 56(12), 2134-2153.

[28] Shah, S. 2006. Motivation, governance and the viability of hybrid forms in open source software development. *Management Science*, 52(7): 1000–1014.

[29] Skinner, B. 1953. *Science and human behavior.* New York: Macmillan.

[30] Stern, S. 2004. Do Scientists Pay to Be Scientists? *Management Science*, 50(6): 835-853.

[31] von Hippel, E. and Krogh, G. 2002. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2): 209–223.

Table 1. Breakdown of Code Contributions: Developer Level

	(1)	(2)	(3)	(4)	(5)
	# Developers	# Contributions	# Receiving Projects	Contributions per Developer (2)/(1)	Contributions per Project (2)/(3)
<u>External Contributors</u>					
Anonymous	NA	18,722	1,939	NA	9.7
Non-members	15,133	32,293	4,071	2.1	7.9
Open	3,211	14,326	2,026	4.5	7.1
Mixed	2,308	25,802	1,784	11.2	14.5
Closed	1,860	12,569	1,356	6.8	9.3
<u>Internal Contributors</u>					
Open	1,184	11,213	581	9.5	19.3
Mixed	1,026	21,769	592	21.2	36.8
Closed	474	9,822	277	20.7	35.5

Notes: This table reports the breakdown of contributions by developer type. Developer types are as follows: Anonymous – developers who do not reveal their identity when making code contributions; Open – developers who are only members of projects with highly restrictive licenses; Closed – developers who are only members of projects with unrestrictive licenses; Mixed – developer who are members of both highly restrictive and unrestrictive projects; Non-members – developers who do not belong to any project, but whose identity is known. Project size is defined as the number of registered members.

Table 2. Distribution of Code Contributions by Developer Type and Receiving Project Characteristics (%)

	Contributing developers				
	(1)	(2)	(3)	(4)	(5)
	Anonymous	Open	Closed	Mixed	Non-members
<u>License Type</u>					
High Open	81.3	67.0	37.6	45.6	57.5
Open	9.9	16.9	26.0	35.0	21.2
Closed	8.8	16.2	36.4	19.4	21.3
<u>Intended Audience</u>					
Developers	8.5	21.7	35.6	40.6	24.1
End-users/Desktop	32.6	24.0	10.2	11.0	42.7
Other	58.9	54.3	54.2	48.4	33.2
<u>Project Size</u>					
1–5	73.1	49.3	38.4	37.9	37.7
6–10	9.2	20.0	17.7	19.4	22.5
11–50	16.9	26.6	36.9	38.1	34.8
> 50	0.8	4.1	7.0	4.6	5.0

Notes: This table reports the distribution of code contributions by developers of different types and receiving project characteristics. We exclude internal contributions – contributions from developers to projects of which they are members.

Table 3. Sorting on License Type

Dependent variable: Number of contributions (Negative Binomial, N=7,705)					
	Developer type				
	(1)	(2)	(3)	(4)	(5)
	Anonymous	Open	Mixed	Closed	Non-members
Receiving project license type:					
Highly Open (HO)	2.19	0.00	0.47	-2.93**	5.08**
	(2.34)		(0.82)	(0.46)	(1.42)
Open (O)	-2.20**	-1.62**	0.99	-2.03**	2.10*
	(0.70)	(0.57)	(1.18)	(0.60)	(1.16)
Closed (C)	-2.95**	-1.96**	0.59	-1.40**	-2.65**
	(0.38)	(0.41)	(0.97)	(0.62)	(1.07)
ln(Number of members)	2.66**	2.69**	2.97**	3.57**	3.22**
	(0.45)	(0.47)	(0.46)	(0.49)	(0.42)
Average # contributions per cell	12.1	9.3	16.7	8.2	21.0
H_0: HO=C	p<0.001	p<0.001	p=0.51	p<0.001	p=0.51

Notes: This table reports the estimated marginal effects (evaluated at the mean) on the interaction terms between the contributing developer type and license type and number of members of the receiving project. The regression includes complete sets of linear dummies for receiving project year of registration, intended audience, operating system, and programming language. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: The coefficient estimates on Intended Audience=0 (p-value<0.001), Programming Language=0 (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate is 3.20 (0.016). The marginal effect of the number of receiving projects in a cell is 0.19 (0.018). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table 4. Sorting on License Type and Intended Audience

Dependent variable: Number of contributions (Negative Binomial, N=7,705)

	Developer type				
	(1)	(2)	(3)	(4)	(5)
	Anonymous	Open	Mixed	Closed	Non-members
Receiving project license type:					
Highly Open (HO)	3.56**	0.00	0.54	-2.95**	5.08**
	(1.14)		(0.70)	(0.35)	(1.23)
Open (O)	-1.03	-1.39**	0.35	-2.30**	2.78*
	(0.67)	(0.45)	(0.83)	(0.45)	(1.23)
Closed (C)	-2.06**	-1.83**	0.27	-1.72**	-3.11**
	(0.50)	(0.36)	(0.76)	(0.52)	(1.19)
Receiving project intended audience:					
Developer Tools	-2.81**	-1.47**	0.46	2.05**	-1.65**
	(0.39)	(0.057)	(0.85)	(0.72)	(0.54)
End Users	-1.56**	0.65	-1.28*	0.14	0.53
	(0.62)	(0.99)	(0.66)	(0.67)	(0.96)
ln(Number of members)	2.47**	2.65**	2.95**	3.58**	3.14**
	(0.36)	(0.030)	(0.31)	(0.33)	(0.32)
Average # contributions per cell	12.1	9.3	16.7	8.2	21.0
H_0: HO=C	p<0.001	p<0.001	p=0.08	p<0.001	p<0.001

Notes: This table reports the estimated marginal effects (evaluated at the mean) on the interaction terms between the contributing developer type and the license type, number of members of the receiving project, and two intended audience dummies – developer tools and end users. The regression includes complete sets of linear dummies for receiving project year of registration, intended audience (excluding the linear dummies for the interacted Developer Tools and End Users categories), operating system, and programming language. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: The coefficient estimates on Programming Language=0 (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate 3.17 (0.064). The marginal effect of the number of receiving projects in a cell is 0.19 (0.012). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table 5. Code Contributions and Project Performance

Dependent variable: ln(1+Number of Project Downloads)

Non-linear Least Squares (N=34,007)

	(1)	(2)	(3)
Elasticity: *Total Contributions Stock*	0.307** (0.007)	0.305** (0.007)	0.297** (0.007)
Marginal Productivity: *Non-member* and anonymous *Contributions* (normalization)		1.000	1.000
Relative Marginal Productivity: *Internal Contributions*		0.344** (0.143)	0.346** (0.012)
Relative Marginal Productivity: *External Contributions*		0.620** (0.138)	
Relative Marginal Productivity: Contributions from *Open and Anonymous Developers*			1.100** (0.183)
Relative Marginal Productivity: Contributions from *Closed Developers*			0.105** (0.030)
Relative Marginal Productivity: Contributions from *Mixed Developers*			0.111** (0.026)
Adjusted-R^2	0.562	0.564	0.564

Notes: This table reports estimates from nonlinear least squares regressions of the log of project downloads on various stocks of contributions, plus dummy variable controls for project characteristics including intended audience, programming language, operating system, and project registration year. Projects that never receive downloads are excluded from the sample. We include a dummy variable that receives the value of one for observations where the number of downloads is zero. Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table 6. Sorting by License Type and Public Resolution

Dependent variable: Number of contributions (Negative Binomial, N=10,755)					
	Developer type				
	(1)	(2)	(3)	(4)	(5)
	Anonymous	Open	Mixed	Closed	Non-members
Project license type:					
Highly Open (HO)	1.39 (1.48)	0.00	0.72 (0.57)	-2.12** (0.30)	3.15** (0.79)
Open (O)	-1.47** (0.44)	-1.12** (0.36)	1.92* (0.93)	-1.18** (0.47)	2.51** (0.99)
Closed (C)	-1.94** (0.29)	-1.37** (0.29)	1.17 (0.77)	-0.74 (0.50)	2.19** (0.71)
Public Resolution	-1.13* (0.62)	4.01** (0.86)	1.83** (0.60)	3.15** (0.90)	2.31** (0.49)
ln(Number of members)	2.90** (0.34)	1.40** (0.22)	1.81** (0.25)	2.06** (0.29)	2.09** (0.21)
Average # contributions per cell	8.7	6.7	12.0	5.8	15.0
H_0: HO=C	p<0.001	p<0.001	p=0.27	p<0.001	p=0.27

Notes: This table reports the estimated marginal effects (evaluated at the mean) of the interaction terms between the contributing developer type and the license type, number of members of the receiving project and a dummy for whether the receiving project publicly reports the outcome of code contributions. Public Resolution takes a value of one for projects that report resolution for at least fifty percent of the contributions they receive. We drop contributions for which a decision has not been made as of the data extraction date. The regression includes complete sets of linear dummies for receiving project year of registration, intended audience, and operating system. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: Public Resolution coefficients equal (p-value<0.001), Intended Audience=0, Programming Language (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate 3.20 (0.14). The marginal effect of the number of receiving projects in a cell is 1.01 (0.076). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table 7. Pattern of Code Contributions by Corporate Sponsorship (%)

	(1)	(2)	(3)	(4)	(5)	(6)
		Developer type				
	Number of projects	Anonymous	Open	Mixed	Closed	Non-members
Not-for-profit	73	85.8	54.8	35.7	21.0	21.3
Corporate Sponsorship	75	14.2	45.2	64.3	79.0	78.7
Total	148	12,134	5,529	11,922	10,658	5,444

Notes: This table reports the pattern of the 45,687 contributions for projects with and without corporate sponsorship.

Table 8. Developer Type and Corporate Sponsorship

Dependent variable: Dummy for Contribution to Corporate Sponsored Project (N=45,687)

	(1)	(2)	(3)	(4)
Dummy for developer type:				
Open: reference category				
Anonymous	-0.362**	-0.288*	-0.253*	-0.014
	(0.147)	(0.135)	(0.123)	(0.089)
Mixed	0.339**	0.278**	0.198**	0.165**
	(0.079)	(0.063)	(0.061)	(0.045)
Closed	0.324**	0.274**	0.227*	0.144*
	(0.081)	(0.084)	(0.106)	(0.063)
Non-members	0.188**	0.118*	0.105	0.111**
	(0.072)	(0.062)	(0.060)	(0.043)
Dummy for license type:				
Highly Open: base category				
Open			0.491**	0.303**
			(0.112)	(0.122)
Closed			0.250**	0.111 (0.171)
			(0.148)	
ln(Number of members)		0.262**	0.083*	0.100**
		(0.053)	(0.043)	(0.037)
Dummies for Intended Audience	No	No	No	Yes
Dummies for Programming Languages	No	No	No	Yes
Dummies for Operating Systems	No	No	No	Yes
R^2	0.212	0.250	0.361	0.443

Notes: This table reports marginal effect estimates from (Probit) regressions for whether a code is contributed to a corporate project, rather than to a non-for-profit project. The dummy equals one if the contribution is to a corporate project, and zero if the contribution is to a non-for-profit project. Analysis is at the contribution level. We include only contributions to projects that we can clearly identify as either corporate or not-for-profit. Standard errors are robust to arbitrary heteroskedasticity and allow for serial correlation through clustering by receiving projects. * denotes statistical significance at the 5% level, and ** at the 1% level.

Table 9. Pattern of Reciprocity

	(1)	(2)	(3)	(4)
Receiving license type:	# of projects	% of projects with reciprocal contributions	% of contributions received by reciprocal projects	% of contributions from projects in (1) that are reciprocal
All	256	4.9	37.0	44.6
Highly Open	142	4.0	20.6	36.0
Open	45	5.5	46.0	39.3
Closed	69	7.5	74.8	50.7

Notes: This table reports the pattern of reciprocity of contributions for projects with different license types. This table includes only projects that receive at least one reciprocal contribution.

Table 10. Determinants of Reciprocity

Dependent variable: Dummy for Reciprocity (N=5,266)

	(1)	(2)	(3)
Dummy for matching on license type:			
All licenses	0.120** (0.027)	0.109** (0.025)	
Highly Open (HO)			0.031 (0.058)
Open (O)			0.061 (0.075)
Closed (C)			0.547** (0.137)
Dummy for matching on intended audiences	0.054** (0.019)	0.045** (0.016)	0.048** (0.017)
Dummy for matching on Programming Language	0.012 (0.023)	-0.037 (0.021)	-0.035 (0.020)
Dummy for matching on Operating Systems	0.049 (0.027)	0.030 (0.022)	0.032 (0.022)
ln(Number of members), receiving		0.003** (0.001)	0.003** (0.001)
ln(Number of members), contributing		-0.001 (0.002)	-0.001 (0.002)
Pseudo-R^2	0.330	0.368	0.375

Notes: This table reports marginal effect estimates from (Probit) regressions for whether a code contribution is reciprocal. The dummy equals one if the contribution is reciprocal and zero otherwise. The dummy variable for matching on license type takes the value one if the contributing and receiving projects have the same license. Other matching dummies are defined similarly. Standard errors are robust to arbitrary heteroskedasticity and allow for serial correlation through clustering by receiving projects. * denotes statistical significance at the 5% level, and ** at the 1% level.

Table 11: Summary of Main Robustness Tests to Address Endogeneity

Source of potential bias	Approach
1. Unobserved project quality	1. Control for *additive* project characteristics using a complete set of categorical dummies and focus on sorting coefficients associated with *interaction effects*. Unobserved project quality induces bias only if it is correlated with these interactions. While higher-quality projects may attract more contributions, this should not systematically affect one contributing type relative to another. 2. Introduce a control for prior cumulative project downloads as a direct (imperfect) measure of project quality.
2. Unobserved developer quality	1. Focus the analysis on sorting coefficients associated with interaction effects and control for *additive* project characteristics using a complete set of categorical dummies. Unobserved developer experience induces bias only if it is correlated with these interactions. While higher-quality developers may make more contributions, this should not systematically vary by receiving project characteristics (especially license type). 2. Introduce a direct measure of developer experience, constructed as the stock of contributions made by a developer prior to the date of the focal contribution. 3. Estimate our baseline specification separately for different developer experience levels.
3. Unobserved technical links between cells (programming languages and operating systems)	1. Re-estimate baseline model at the project (rather than cell) level, which allows us to control for both contributing and receiving project characteristics. 2. Introduce dummy variables that capture matching between receiving and contributing project characteristics. 3. Calculate a continuous measure of technological similarity between receiving and contributing project characteristics, in the spirit of knowledge spillover measures (Jaffe, 1986) and use it as an additional control variable.

A. Data Appendix

We developed specialized web-crawler software to extract information from the SourceForge website (a procedure that takes about a week to complete). We used two different versions of the software to implement the extraction (due to changes in the website format): November 2005 and September 2010. In each extraction we retrieved information for all projects listed on SourceForge, and all relevant information for each project. This repetition allows us to check the consistency of information within and across projects over time.

The 2005 extraction covers 77,813 projects, while our final 2010 extraction (on which the econometric analysis is based) includes 215,072 projects, an increase of 76.4%. In the 2005 extraction, 4,086 projects receive at least one internal or external contribution, which is 5.3% of all registered projects. In the 2010 extraction, the corresponding figure is 3.5% of all projects. There was also a shift in the distribution of projects across license types, with closed licenses more heavily represented in the later sample. The composition of the 2010 sample is: 46.7% Highly Open, 9.8% Open, and 43.5% Closed. The composition in 2005 is: 68.7% Highly Open, 15.2% Open, 13.5% Closed, 2.0% Public Domain, and 0.7% unidentified. This shift toward projects with closed licenses likely reflects the increasing involvement of corporate sponsorship.

The number of developers registered in SourceForge also increased sharply over time, from 113,191 in 2005 to 211,711 in 2010, an increase of 87.0%. Of those registered in 2005, 13.5% of developers made at least one contribution in the 2000–2005 period. Of the developers registered in 2010, 12.3% made at least one contribution in the 2000–2010 period.

We run checks for two distinct aspects of data consistency: 1) the risk that contributions from existing projects are dropped by SourceForge (*attrition of contributions*), and 2) the risk that projects are dropped over time by SourceForge (*attrition of projects*). Given that our econometric analysis focuses on the pattern of contributions, attrition of contributions is the more serious concern as it is associated with incompleteness of the history of the object of interest. The attrition may vary over time and thus can potentially bias our results (e.g., younger projects have a more complete history file, and systematically have a different license type than older projects). In cases of project attrition, however, we do not observe these projects in the 2010 extraction but, for those that are present, we have complete historical information on contributions.

Before turning to the details, we summarize our findings briefly as follows. First, we find no evidence of contribution attrition. For the projects in the 2010 extraction, the history files are complete (i.e., there are no contributions registered for projects in 2005 that are missing in 2010). Second, we do find evidence of project attrition—some projects active before 2010 were dropped from SourceForge and do not appear in the 2010 extraction. However, as explained below, we do not think that any systematic bias in our econometric analysis is likely to be caused by this attrition.

We begin with project attrition between the two data extractions. Of the 77,813 projects registered on SourceForge in 2005, 67% also appear in 2010. Of the active projects (those that receive at least one contribution), the corresponding percentage is 83%. There are two main reasons why projects are dropped from the sample over time. First, some projects were removed from SourceForge (for example, *Arkipel Project* was active in 2005 but does not appear in SourceForge in 2010). Second, some projects change their names, which makes it difficult to identify them using automated name-matching. These projects are not dropped from the data, and should not cause any bias in a single cross-sectional extraction. An example of a name-changing project is *ActionCube* (its 2005 name), which appears under the name *AssaultCube* in 2010.

We turn next to contribution attrition—the extent to which code contributions are dropped from registered projects over time. The 2005 data extraction includes 51,545 contributions, 85% of which also appear in the 2010 extraction. Nearly all of the contributions that do not appear in the 2010 extraction are missing because the projects themselves have been dropped. Only 200 contributions that appear in the 2005 data and belong to projects that are included in the 2010 data extraction are missing from the 2010 data. That is, while some projects are dropped from the sample over time, observing a project in a given year provides very accurate information on the history of code submission. We conclude that there is no evidence of any systematic time-inconsistency with regard to information on contributions.

There are two main reasons why contributions are dropped through project attrition. First, several large projects were de-activated and their activity was transferred to other websites. For example, 55% of the dropped contributions belong to *Python*, which moved to http://www.python.org/. However, during the period that *Python* appears on SourceForge, we observe its complete historical and current contributions. Second, some projects, such as *Scons* and *Jython*, do not provide access to their history in 2010, but did provide access in 2005 (e.g., the site for *Scons*, http://sourceforge.net/projects/scons/develop, does not provide information on submissions).

We also checked whether characteristics of the projects change over time. We turn first to the project license type. Of the projects in 2010 that were also registered before 2005, 93.3% have the same license type recorded at both dates. A similar pattern holds for other project characteristics—the figures for intended audience and programming language are 94.1% and 83.8%, respectively.

Definition of programming language categories

Projects fall under one of 70 different programming languages. Based on discussions with software developers, we group these languages into five broad categories: object-oriented, imperative, scriptive, dynamic, and other (see Data Appendix for details). We also include a separate category for projects that do not specify their programming languages (14% of projects). We use five categories in the empirical analysis. The programming languages in each are as follows:

Object-oriented: Java, C++, Smalltalk, Visual Basic NET, C#, Object Pascal, Delphi/Kylix, Visual Basic, Ada, D, Groovy, PL/SQL, AspectJ, COBOL, JSP, LPC, REALbasic, Visual FoxPro, Zope, OCaml, and Simula

Imperative: C, Fortran, Standard ML, PROGRESS, and Pascal

Scriptive: JavaScript, PHP, Tcl, Rexx, ActionScript, Emacs-Lisp, VBScript, Cold Fusion, AWK, and AppleScript

Dynamic: Perl, Python, Dylan, Erlang, Forth, Lisp, Logo, Scheme, Lua, and Modula

Objective: C, Ruby, ASP.NET, Common Lisp, Pike, Prolog, Eiffel, REBOL, and Euler

Other: Assembly, UnixShell, ASP, Haskell, APL, MATLAB, BASIC, XBasic, Euphoria, IDL, LabVIEW, XSL, and VHDL/Veril

Definition of operating system categories

Each project is conducted on one or more operating system, which is the platform on which the program runs. We use four groups of operating systems: Microsoft, Open Source Independent, POSIX, and Mixed (see Data Appendix for details). We also include a separate category for projects that do not specify their operating system platform (20% of projects). We use four categories of operating systems in the empirical analysis. Using Lerner and Tirole (2002), Wikipedia, and http://osapa.org/wiki/index.php/SF/Freshmean Trove, we group the operating systems into the following categories:

Microsoft: all of Microsoft's operating systems (e.g., MS-DOS and WinXP)

POSIX: Linux, Solaris, and BSD

Open Source independent: any independent open-source operating system

Mixed: any software that operates on more than one operating system

Project performance

Beginning in 2006, SourceForge provides information that can be used to study projects' performance. We focus on projects that received at least one contribution between 2001 and 2010. We were able to match 66% of these projects to those used in our analysis of sorting. For the others, either the name of the project changed or there was no available performance data. We measure project performance by the number of times the project is downloaded. This is a good indicator of the extent to which the project diffuses among potential users and developers. We also used two other measures: the number of web hits the project gets on the SourceForge site, and the number of the project's web pages that are viewed by users registered on SourceForge (the latter is more likely to capture diffusion among software developers). The qualitative results with these performance measures are similar to those reported in this section.

Table A1. Linear Coefficient Estimates for Tables 3, 4, and 6

Dependent variable: Number of contributions			
	(1)	(2)	(3)
	Table 3	Table 4	Table 6
Dummy for programming languages:			
Dynamic	2.40**	2.21**	0.108
	(0.99)	(0.65)	(0.58)
Imperative	7.10**	6.75**	1.93**
	(1.83)	(0.99)	(0.81)
Object-Oriented	4.67**	4.52**	1.74**
	(1.12)	(0.70)	(0.63)
Other	-1.02	-1.02	-2.02**
	(1.01)	(0.62)	(0.64)
Scriptive	4.12**	3.86**	2.71**
	(1.25)	(0.79)	(0.89)
Dummy for operating systems:			
Broad	4.34**	4.33**	2.72**
	(0.75)	(0.53)	(0.56)
Microsoft	2.11*	2.12**	2.34**
	(0.93)	(0.61)	(0.98)
OS Independent	3.13**	3.14**	1.68**
	(0.71)	(0.51)	(0.55)
Posix	4.66**	4.63**	3.44*
	(1.00)	(0.60)	(0.81)

Notes: This table reports the coefficient estimates of the linear controls for programming languages and operating systems which are included in tables 3,4 and 6 (but are not reported in these tables for brevity). The baseline category for programming languages and operating systems is "unknown". * denotes statistical significance at the 5% level, and ** at the 1% level.

Table A2. Contribution Distribution by Developer Experience to Different License Types (%)

	Contributing developers			
	(1)	(2)	(3)	(4)
	Open	Mixed	Closed	Non-member
License Type				
	All developers			
Highly Open	81.7	46.5	19.3	57.5
Open	9.3	33.0	14.6	21.2
Closed	9.0	20.5	66.0	21.3
	Highly-experienced developers (highest quartile)			
Highly Open	95.3	47.0	8.0	98.4
Open	3.6	38.0	5.7	0.3
Closed	1.1	15.1	86.4	1.3
	Medium-experienced developers (second and third quartiles)			
Highly Open	81.4	46.3	17.4	57.6
Open	9.6	29.3	18.8	22.7
Closed	9.0	24.5	63.7	19.7
	Inexperienced developers (lowest quartile)			
Highly Open	73.0	46.0	36.3	56.7
Open	12.7	30.8	18.4	20.8
Closed	14.4	23.2	45.3	22.4

Notes: Experience is defined as the stock of contributions made by a developer up to the date of the focal contribution.

Table A3.a. Controlling for Developer Experience

Dependent variable: Number of contributions (Negative Binomial, N=7,705)

	Developer type			
	(1)	(2)	(3)	(4)
	Open	Mixed	Closed	Non-members
Project license type:				
Highly Open (HO)	0.00	-0.30	-2.35**	5.46**
		(0.43)	(0.28)	(0.94)
Open (O)	-1.16**	-0.08	-1.91**	3.75**
	(0.34)	(0.55)	(0.33)	(1.02)
Closed (C)	-1.24**	0.14	-0.99**	3.95**
	(0.31)	(0.54)	(0.39)	(0.88)
ln(Number of members)	2.33**	1.72**	2.20**	2.74**
	(0.27)	(0.23)	(0.27)	(0.27)
Developer stock of prior source code contributions	0.25**			
	(0.04)			
Average # contributions per cell	9.3	16.7	8.2	21.0
H_0: HO=C	p<0.001	p=0.29	p<0.001	p<0.001

Notes: This table reports the estimated marginal effects (evaluated at the mean) on the interaction terms between the contributing developer type and the license type, when controlling for developer experience. Experience is defined as the stock of contributions made by a developer up to the date of the focal contribution. Anonymous developers are excluded because we cannot calculate experience for developers that do not reveal their identity. The regression includes complete sets of linear dummies for receiving project year of registration, intended audience, operating system, and programming language. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: Intended Audience=0 (p-value<0.001), Programming Language=0 (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate 2.35 (0.08). The marginal effect of the number of potential contributing projects is 0.14 (0.01). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table A3.b. Removing Inexperienced Developers

Dependent variable: Number of contributions (Negative Binomial, N=6,164)

	Developer type			
	(1)	(2)	(3)	(4)
	Open	Mixed	Closed	Non-members
Project license type:				
Highly Open (HO)	0.00	1.37 (0.76)	-1.54** (0.39)	-0.30 (0.53)
Open (O)	-1.38** (0.33)	1.33 (0.96)	-1.02* (0.49)	-1.57** (0.37)
Closed (C)	-1.32** (0.29)	1.17 (0.82)	-0.63 (0.50)	-1.38** (0.36)
ln(Number of members)	1.55** (0.29)	1.44** (0.27)	2.00** (0.31)	2.47** (0.33)
Average # contributions per cell	5.9	14.3	6.3	7.0
H_0: HO=C	p<0.001	p=0.72	p<0.05	p<0.001

Notes: This table reports the estimated marginal effects (evaluated at the mean) on the interaction terms between the contributing developer type and the license type when removing contributions by inexperienced developers (lowest experience quartile). Anonymous developers are excluded because we cannot calculate experience for developers that do not reveal their identity. The regression includes complete sets of linear dummies for receiving project year of registration, intended audience, operating system, and programming language. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: Intended Audience=0 (p-value<0.001), Programming Language=0 (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate 2.35 (0.08). The marginal effect of the number of potential contributing projects is 0.14 (0.01). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table A4. Project Level Estimation: Controlling for Contributing Project Characteristics

Dependent variable: Number of contributions (Negative Binomial, N=374,808)

	(1) Baseline Project-level			(2) Same Programming Language Dummy			(3) Complete set of Same Programming Language Dummies			(4) Programming Language Distance (Jaffe, 1986)			(5) Complete Set of Programming Language Interactions		
	Open	Mixed	Closed	Open	Mixed	Closed	Open	Mixed	Closed	Open	Mixed	Closed	Open	Mix	Closed
Highly Open (HO)		0.03 (0.11)	-1.41** (0.11)		-0.01 (0.11)	-1.40** (0.11)		0.04 (0.11)	-1.40** (0.11)		-0.00 (0.14)	-1.52** (0.14)		0.03 (0.11)	-1.49** (0.11)
Open (O)	-0.35** (0.11)	-0.06 (0.13)	-1.14** (0.13)	-0.39** (0.11)	-0.03 (0.13)	-1.16** (0.13)	-0.42** (0.11)	-0.02 (0.13)	-1.18** (0.13)	-0.46** (0.14)	-0.03 (0.17)	-1.27** (0.17)	-0.36** (0.11)	-0.05 (0.13)	-1.18** (0.13)
Closed (C)	-0.59** (0.10)	-0.11 (0.13)	-0.54** (0.13)	-0.56** (0.10)	-0.08 (0.12)	-0.57** (0.13)	-0.59** (0.10)	-0.09 (0.12)	-0.56** (0.13)	-0.56** (0.13)	-0.05 (0.16)	-0.72** (0.17)	-0.56** (0.10)	-0.05 (0.13)	-0.65** (0.13)
ln(*Number of members*)	0.72** (0.06)	0.61** (0.06)	0.61** (0.07)	0.76** (0.06)	0.64** (0.06)	0.62** (0.07)	0.79** (0.06)	0.64** (0.06)	0.65** (0.07)	0.81** (0.08)	0.67** (0.07)	0.63** (0.08)	0.72** (0.06)	0.62** (0.06)	0.62** (0.07)
Dummy for Same Programming Language					0.90** (0.06)									2.07** (0.29)	
Dummy for Both Dynamic								0.96** (0.14)							
Dummy for Both Imperative								0.56** (0.14)							
Dummy for Both Object-Oriented								0.90** (0.12)							
Dummy for Both Other								0.36 (0.33)							
Programming Languages Distance Measure											3.63** (0.83)				
Number of contributing projects		0.001** (0.0001)			0.001** (0.0001)			0.001** (0.0001)			0.001** (0.0001)			0.001** (0.0001)	
Number of receiving projects		0.05** (0.001)			0.05** (0.001)			0.05** (0.001)			0.05** (0.001)			0.05** (0.001)	
Pseudo R^2		0.029			0.031			0.030			0.030			0.031	

Note: This table reports the estimated marginal effects (evaluated at the mean) of a model where "cells" are defined at the level of contributing and receiving project characteristics. The regressions include complete sets of linear dummies for receiving project year of registration which we do not report for space considerations. Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table A4 *(Cont'd)* **. Project Level Estimation: Controlling for Contributing Project Characteristics**

Dependent variable: Number of contributions (Negative Binomial, N=374,808)

	(1) N=374,808	(2) N=374,808	(3) N=374,808	(4) N=298,740	(5) N=374,808
Contributing Projects Characteristics					
Intended audience dummies					
End Users	0.05 (0.08)	0.07 (0.07)	0.06 (0.08)	0.10 (0.10)	0.05 (0.08)
Other	-0.21** (0.07)	-0.20** (0.07)	-0.21** (0.07)	-0.23** (0.09)	-0.22** (0.07)
System Administrators	-0.64** (0.09)	-0.62** (0.08)	-0.59** (0.09)	-0.70** (0.11)	-0.67** (0.09)
Mixed	-0.91** (0.09)	-0.83** (0.09)	-0.83** (0.09)	-0.70** (0.11)	-0.85** (0.09)
Programming language dummies					
Dynamic	0.71** (0.07)	0.74** (0.07)	0.48** (0.08)	0.17 (0.19)	
Imperative	0.56** (0.08)	0.61** (0.07)	0.43** (0.08)	0.10 (0.20)	
Object-Oriented	0.87** (0.09)	0.79** (0.09)	0.52** (0.10)	0.31 (0.20)	
Other	-1.99** (0.09)	-1.75** (0.08)	-1.97** (0.09)	-1.29** (0.14)	
Operating systems dummies					
Broad	-0.24** (0.07)	-0.19** (0.07)	-0.21** (0.07)	-0.11 (0.08)	-0.19** (0.07)
Microsoft	-1.35** (0.08)	-1.27** (0.08)	-1.28** (0.08)	-1.29** (0.10)	-1.26** (0.08)
OS Independent	-0.46** (0.07)	-0.44** (0.07)	-0.46** (0.07)	-0.41** (0.09)	-0.43** (0.07)
Receiving Projects Characteristics					
Intended audience dummies					
End Users	-0.27** (0.07)	-0.18* (0.07)	-0.21** (0.07)	-0.08 (0.09)	-0.17* (0.07)
Other	-0.55** (0.08)	-0.47** (0.08)	-0.51** (0.08)	-0.29** (0.10)	-0.45** (0.08)
System Administrators	-0.01 (0.10)	0.09 (0.10)	0.03 (0.10)	0.29* (0.12)	0.11 (0.10)
Mixed	-0.11 (0.07)	-0.06 (0.07)	-0.08 (0.07)	0.01 (0.08)	-0.04 (0.07)
Programming language dummies					
Dynamic	-0.12 (0.08)	-0.04 (0.08)	-0.34** (0.08)	-0.58** (0.19)	
Imperative	0.29** (0.08)	0.42** (0.08)	0.21* (0.09)	-0.08 (0.21)	
Object-Oriented	0.01 (0.08)	0.13 (0.08)	-0.14 (0.08)	-0.39 (0.21)	
Other	-1.32** (0.14)	-1.15** (0.14)	-1.36** (0.15)	-0.56** (0.20)	
Operating systems dummies					
Broad	0.19** (0.06)	0.23** (0.06)	0.23** (0.06)	0.35** (0.08)	0.28** (0.06)
Microsoft	-0.32** (0.09)	-0.38** (0.09)	-0.37** (0.09)	-0.40** (0.12)	-0.31** (0.09)
OS Independent	-0.16* (0.07)	-0.18* (0.07)	-0.15* (0.07)	-0.18 (0.09)	-0.21** (0.07)
Pseudo R^2	0.029	0.031	0.030	0.030	0.031

Table A4 (Cont'd). Project Level Estimation: Controlling for Contributing Project Characteristics

Dependent variable: Number of contributions (Negative Binomial, N=374,808)

	(5)
Programming Language Interaction Dummies (Contributing × Receiving)	
Dynamic × Dynamic	0.30 (0.16)
Dynamic × Imperative	2.02** (0.28)
Dynamic × Object-Oriented	1.64** (0.29)
Dynamic × Other	0.27 (0.37)
Dynamic × Scriptive	1.42** (0.29)
Imperative × Dynamic	1.59** (0.28)
Imperative × Imperative	0.37* (0.16)
Imperative × Object-Oriented	1.53** (0.28)
Imperative × Other	-0.14 (0.37)
Imperative × Scriptive	1.44** (0.29)
Object-Oriented × Dynamic	1.62** (0.29)
Object-Oriented × Imperative	2.11** (0.29)
Object-Oriented × Object-Oriented	0.44** (0.16)
Object-Oriented × Other	0.53 (0.36)
Object-Oriented × Scriptive	1.37** (0.29)
Other × Dynamic	-1.47** (0.29)
Other × Imperative	-0.45 (0.29)
Other × Object-Oriented	-0.58* (0.28)
Other × Other	-3.88** (0.33)
Other × Scriptive	-1.23** (0.30)
Scriptive × Dynamic	0.78** (0.29)
Scriptive × Imperative	1.05** (0.28)
Scriptive × Object-Oriented	0.60* (0.28)
Pseudo R^2	0.031

Table A5. Robustness Checks: Interaction Between Number of Potentially Receiving Projects and Developer Type

Dependent variable: Number of contributions (Negative Binomial, N=7,705)

	Developer type				
	(1)	(1)	(2)	(3)	(4)
	Anonymous	Open	Mixed	Closed	Non-members
Project license type:					
Highly Open (HO)	2.86 (2.89)	0.00	0.19 (0.92)	-2.97** (0.52)	5.18** (1.66)
Open (O)	-2.08** (0.76)	-1.62** (0.57)	0.72 (1.14)	-2.07** (0.63)	2.15 (1.25)
Closed (C)	-2.85** (0.42)	-1.96** (0.42)	0.40 (1.02)	-1.44* (0.65)	2.69* (1.17)
ln(Number of members)	2.79** (0.46)	2.68** (0.47)	2.83** (0.46)	3.52** (0.49)	3.22** (0.42)
Number of potential receiving projects × Developer type:	0.14** (0.04)	0.19** (0.03)	0.22** (0.04)	0.20** (0.04)	0.18** (0.03)
Average # contributions per cell	12.1	9.3	16.7	8.2	21.0
H_0: HO=C	p<0.001	p<0.001	p=0.78	p<0.01	p<0.01

Notes: This table reports the estimated marginal effects (evaluated at the mean) on the interaction terms between the contributing developer type and the license type, when controlling for interactions between the number of potential receiving projects in a cell and developer type. The regression includes complete sets of linear dummies for receiving project year of registration, intended audience, operating system, and programming language. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: Intended Audience=0 (p-value<0.001), Programming Language=0 (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate 3.20 (0.16). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

TABLE A6. Controlling for Number of Contributing Developers

Dependent variable: Number of contributions (Negative Binomial, N=6,164)				
	Developer type			
	(1)	(2)	(3)	(4)
	Open	Mixed	Closed	Non-members
Project license type:				
Highly Open (HO)	0.00	0.64 (0.59)	-2.23** (0.41)	1.52** (0.61)
Open (O)	-1.55** (0.37)	0.09 (0.65)	-1.88** (0.45)	0.33 (0.55)
Closed (C)	-1.58** (0.30)	0.45 (0.66)	-1.46** (0.45)	0.98 (0.59)
ln(Number of members)	1.26** (0.27)	1.74** (0.28)	2.57** (0.36)	1.23** (0.27)
Number of contributing developers in a cell	0.33** (0.04)			
Average # contributions per cell	9.3	16.7	8.2	21.0
H_0: HO=C	p<0.001	p=0.72	p<0.05	p=0.16

Notes: This table reports the estimated marginal effects (evaluated at the mean) on the interaction terms between the contributing developer type and the license type when controlling for the number of contributing developers in a cell. We exclude anonymous developers because for them we cannot determine the number of unique developers. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: Intended Audience=0 (p-value<0.001), Programming Language=0 (p-value<0.001), Operating Systems=0 (p-value<0.001), Size coefficients are equal (p-value<0.001). Over-dispersion parameter estimate 2.38 (0.09). The marginal effect of the number of potential contributing projects is 0.14 (0.01). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

Table A7. Zero-Inflated Negative Binomial Estimation

Dependent variable: Number of contributions (N=7,705)					
			Developer type		
	(1)	(1)	(2)	(3)	(4)
	Anonymous	Open	Mixed	Closed	Non-members
Project license type:					
Highly Open (HO)	2.19**	0.00	-0.47	-2.93**	5.08**
	(0.86)		(0.62)	(0.32)	(1.11)
Open (O)	-2.20**	-1.62**	0.99	-2.03**	2.10*
	(0.44)	(0.42)	(0.86)	(0.45)	(1.01)
Closed (C)	-2.95**	-1.96**	0.59	-1.40**	2.65**
	(0.33)	(0.35)	(0.75)	(0.51)	(1.02)
ln(Number of members)	2.66**	2.69**	2.97**	3.57**	3.22**
	(0.36)	(0.31)	(0.32)	(0.34)	(0.31)
Average # contributions per cell	12.1	9.3	16.7	8.2	21.0
H_0: HO=C	p<0.001	p<0.001	p=0.84	p<0.001	p<0.001

Notes: This table reports the robustness of our results for Zero-Inflated Negative Binomial (ZINB) estimation. The Vuong test for ZINB vs. Negative Binomal is -1.82 (p-value=0.97). The regression includes complete sets of linear dummies for receiving project year of registration, intended audience, operating system, and programming language. We also include a linear control for the number of projects in the cell. Over-dispersion parameter estimate 3.20 (0.07). The marginal effect of the number of potential contributing projects is 0.14 (0.01). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.