CENTRE for ECONOMIC PERFORMANCE

CEP Discussion Paper No 893

October 2008
(Revised January 2012)

Motivation and Sorting in Open Source Software Innovation

Sharon Belenzon and Mark Schankerman

LSE THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

E·S·R·C ECONOMIC & SOCIAL RESEARCH COUNCIL

Abstract
This paper studies the role of intrinsic motivation, reputation, and reciprocity in driving open source software innovation. Unlike previous literature based on survey data, we exploit the observed pattern of contributions - the .revealed preference. of developers - to infer the underlying incentives driving the decision to contribute source code. Using detailed information on code contributions and project membership, we classify software developers into distinct types and study how contributions from each developer type vary according to the open source license type and other project characteristics. We find that developers strongly sort by license type, project size, and corporate sponsorship, and that reciprocity is important only for a small subset of projects. We also show that contributions have a substantial impact on the performance of open source projects.

Keywords: Open source software, innovation, incentives, intrinsic motivation, motivated agents, reputation, reciprocity
JEL Classifications: L14, L17, L41, O31 and O32

This paper was produced as part of the Centre's Productivity and Innovation Programme. The Centre for Economic Performance is financed by the Economic and Social Research Council.

Acknowledgements
We would like to thank Jacques Cremer, Marc Ivaldi, Josh Lerner, Jean Tirole and seminar participants at the NBER for constructive comments on an earlier version of the paper. We gratefully acknowledge the financial support from the British Academy and the Centre for Economic Performance at the London School of Economics, which made it possible for us to construct the data set used in this project. We thank Hadar Gafni for excellent research assistance during the project.

Sharon Belenzon is an Associate of the Centre for Economic Performance, London School of Economics. He is also an Assistant Professor of Strategy at the Fuqua School of Business, Duke University, USA. Mark Schankerman is a Research Associate of the Centre for Economic Performance and Professor of Economics, London School of Economics and the first James and Pamela Muzzy Chair in Entrepreneurship at the University of Arizona's Eller College of Management.

Published by
Centre for Economic Performance
London School of Economics and Political Science
Houghton Street
London WC2A 2AE

ISBN 978-0-85328-300-3

1. Introduction

This paper studies how motivations and sorting behavior affect innovation in open source software. In particular, we study the decision of independent developers to contribute code to different types of open source projects, and the impact of these contributions on project performance.

In open source software (OSS), the source code is available for public use and development under specific conditions which depend on the license governing the project. Programmers who contribute to open source are typically unpaid, though corporate sponsorship and financing have increased sharply in recent years. This raises an important question: How is open source innovation sustained in the face of free-rider problems and in the absence of direct monetary compensation? This is a central issue not only for the software sector, but also for other areas in which "open commons" production has been proposed, including databases, biotechnology, and nanotechnology.[1]

Management and economic scholars have studied this question and have proposed four broad types of theoretical explanations for the paradox. First, developers may be intrinsically motivated to contribute by their strong identification with the "ideology" underlying the open source movement (Raymond, 2001).[2] Second, code contribution and active participation in the OS community may enhance reputation via greater peer recognition (Raymond, 2001) or commercial rewards in the labor market (Lerner and Tirole, 2001, 2002; Johnson, 2002, 2004). Third, developers may expect to gain later 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). For such "hobbyists," the marginal utility of effort may be positive, at least over some range of effort (Kreps, 1997; Glazer, 2004).

These explanations involve varying levels of intrinsic and extrinsic motivation, which have been treated empirically in a number of studies within an OS setting. Though these studies have provided evidence for diverse motivations, they share two important limitations. First, prior works are typically based on relatively small samples of contributors to open source projects (Haruvy, Wu, and Chakravarty, 2003; Hertel, Krishnan, and Slaughter, 2003). The second, and more important limitation, is that they rely exclusively on what programmers say their motivations are, without any direct way to corroborate these "announced preferences." [3]

The strategy of this paper is to utilize revealed preferences, as captured by observed code contributions,

[1] Good general discussions of open source in software and other areas are available in Lerner and Tirole (2005) and Maurer and Scotchmer (2006). For a recent book arguing the case for open source in biotechnology, see Hope (2008), and on nanotechnology, Bruns (2001).

[2] The original open source license that embodies this view is the general purpose license (GPL), which requires that the source code and any subsequent code that builds on it or embodies it must remain open source.

[3] One notable exception is Hann, Roberts, Slaughter, and Fielding (2004), who test the labor market hypothesis of Lerner and Tirole (2002) by studying the relationship between wages of developers and their contributions to the Apache OSS project. They find that wages are related to contributors ranking by the Apache Foundation, but not to the volume of their contributions, suggesting that contributions may be motivated more by labor market signaling than by human capital accumulation.

in order to try to quantify how reputation, reciprocity, utility, and intrinsic motivations each drive open source innovation. Our empirical analysis of code contributions to OSS projects is based on a large-scale dataset with detailed information on both the contributing and receiving projects. Each contribution includes a dyad consisting of a *contributing project,* where the contributor is a registered member, and a *receiving project* to which the code is submitted. The distinction between the contributing and receiving projects is central to our empirical analysis of sorting, 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 the fact that projects vary in the degree to which their licenses are "open," in the spirit of open source. 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 for the source code to be used in proprietary ways that limit terms of use (for discussion, see Lerner and Schankerman, 2010). For ease of exposition, we refer to the latter as "closed" projects.

To study this sorting, we exploit characteristics of the projects such as the 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 license which govern the contributing project with which they are affiliated: *open, closed, mixed,* and *anonymous*. We investigate how the pattern of contributions from each developer type varies across the various characteristics of the contributing-receiving project 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 incentives.

Our econometric approach is to aggregate 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 explore how the likelihood of code contribution varies by the likely motivation of the contributing developer, and the expected potential intrinsic or extrinsic reward that is associated with the specific bundle of receiving project characteristics. The key identification assumption in this paper is that the characteristics of contributing and receiving projects (such as license type) are exogenous with respect to the decisions of individual developers to contribute. The main identification concern with this approach would be unobserved developer or project heterogeneity which may be correlated both with the level of contributions and some of the characteristics of the contributing or receiving project. However, because our focus is on the *interactions* between the contributing developer type and project characteristics (not on the level effects of these characteristics), such heterogeneity would induce bias only if it is correlated with these interactions, which is much less likely. For example, code contributions are likely to be affected by the unobserved quality of the contributing developer. However,

while this unobserved quality would affect her *level* of contributions, there is no reason to expect that it would affect the *distribution* of her contributions in relation to project characteristics.

The empirical findings in this paper show that developers seem to strongly sort on a variety of observed project characteristics. We interpret this as consistent with the view that software developers are heterogeneous with respect to their motivations. First, we find assortative matching on the project license type. Open contributors almost exclusively contribute to projects with open licenses, indicating an important role for "motivated agents" – developers dedicated to the ideology of the open source movement. Closed contributors primarily contribute to projects with closed (more commercial) licenses. If developers understand that such matching occurs, this finding is consistent with the reputation incentive – they go where reputation gains are most likely to be obtained.

Second, contributors from closed projects are more likely to contribute to larger projects and to those that are sponsored by corporations. This evidence supports the view that labor market reputation (career concern) plays an important role, as emphasized by Lerner and Tirole (2002). At the same time, however, we also find that, to a lesser extent, the size of the receiving project matters for other developer types. This indicates that the peer recognition motive also plays a role.

Third, open contributors are much more likely to contribute to projects aimed at end users (e.g., computer games), while closed contributors target developer-oriented (e.g., programming tool) projects. This is potentially important since the development of software tools is the "basic research" that is critical to the long-run sustainability of the sector. These findings also suggest that open source development by intrinsically motivated agents is more of a substitute for proprietary software innovation on the end-product side.[4] All these three findings on sorting behavior are robust to various empirical specifications including a wide range of control variables.

Fourth, we find that reciprocity plays a limited role in sustaining innovation. Developers are more likely to contribute to projects from which they have previously received contributions, controlling for various observed project characteristics. Reciprocity is more common among closed (commercially oriented) developers than for open developers (motivated agents). This suggests that reciprocity is associated more with building reputations than with intrinsic motivation. Relatedly, the vast majority of projects exhibit no reciprocal contributions. But while we observe reciprocity in only a small percentage of projects, reciprocal contributions account for a large fraction of the contributions for those projects.

Finally, we empirically examine the relationship between project performance and the contributions received from project-member developers versus non-members. We show that both internal (member) and external contributions strongly affect project performance, but the impact of contributions by non-members is much larger. Interestingly, while sorting strongly affects the flow of contributions from different

[4] It is important to note that our data set does not include the Linux open source project, which is a widely used operating system.

developer types to projects, we find no differences in the impacts of these different (external) contributions on project performance. That is, sorting behavior by developers affects the quantity, but not the quality, of contributions.

Our findings that motivations are heterogenous, and induce sorting behavior, should also be relevant for theories about employment contract design beyond the OSS context. Principal agency theory, with its focus on extrinsic rewards, has been the dominant paradigm for thinking about employment contracts. But in the last ten years a number of studies have begun to explore the interaction between intrinsic and extrinsic motivations and its implications for optimal contracting.[5] This literature shows that optimal incentives may depend strongly on the form and heterogeneity of worker motivation. When intrinsic motivation is strong (e.g., in the academic and NGO sectors), low-powered incentives may be more efficient for the principal, and extrinsic incentives may even crowd out intrinsic motivation.[6] In addition, the sorting behavior that can arise from heterogeneous motivations may need to be taken into account for econometric studies of contract design (Ackerberg and Botticini, 2002).

In this paper we treat the choice of project license as exogenous. Lerner and Tirole (2002) model the choice of open source license, arguing that the relevant trade-off is between greater proprietary control with the more commercial, closed licenses and a potentially greater pool of contributors with more open licenses.[7] The sorting effect of the project license plays a central role in their theory, and our results provide econometric evidence supporting their perspective. We leave for future work the task of incorporating both the choice of license and the resulting sorting effects into one empirical framework.

The paper is organized as follows: Section 2 sets out the theoretical framework and discusses various developer motivations for open source innovation, which generates the empirical hypotheses we then investigate. Section 3 describes the data set and the key variables used in the empirical analysis. Section 4 presents the econometric framework for studying sorting behavior of developers. Section 5 presents descriptive statistics on the main features of sorting by developers. In Section 6 we present the empirical results on sorting and discuss their implications. Section 7 presents evidence on the impact of contributions from different sources on the performance of open source projects. Brief concluding remarks follow.

2. Theoretical Background and Hypotheses

There are two broad classes of theories about what motivates programmers to contribute to open source projects: intrinsic and extrinsic motivation. The concept of extrinsic motivation was first introduced by Skinner (1953) and further developed by Deci (1971), who defined extrinsically motivated behaviors

[5] This work has focused primarily on government and non-profit organizations, where it is believed that workers may be either intrinsically motivated or more strongly motivated agents in the sense that their preferences are aligned with the employer's mission. Leading examples include Frey (1997), Kreps (1997), Francois (2000), Murdock (2002), Bénabou and Tirole (2003), Besley and Ghatak (2005), Delfgauuw and Dur (2007, 2008), and Prendergast (2008).

[6] For a empirical examples of crowding out effects, see Frey (1997) and Frey and Oberholzer-Gee (1997).

[7] We paraphrase here, using "open" and "closed" according to our present working definition.

as those that are seen by actors as means to an end. When thusly motivated, actions are performed merely in order to attain some separable outcome to which external rewards are attached (such as status, approval, or monetary compensation). This is what most people in economics and management would associate with "incentives." [8] In the absence of a commensurably meaningful reward the action would not be performed. In contrast, an individual is considered intrinsically motivated to act if she "performs an activity for no apparent reward except the activity itself" (Deci, 1972).

Within psychology, the study of intrinsic motivations remains dominated by the work of Deci and Ryan, who incorporated the concepts of intrinsic motivations into their self-determination theory (SDT). This theory explores what intrinsic motivation consists of, above and beyond being merely the opposite of the more intuitive extrinsic motivation. As well, intrinsic motivation is here construed as having its own type of rewards – hard to observe though they may be. Thus, SDT builds on the premise that some actions in and of themselves can satisfy the three crucial psychological needs of competence, autonomy, and relatedness, leading to "an energizing state that, if satisfied, conduces toward health and well-being, but, if not satisfied, contributes to pathology and ill-being" (Ryan and Deci, 2000). The need for competence leads people to seek challenges that are optimal for their ability level and to attempt to both maintain and enhance those skills and abilities (Ryan, 1995). The construct of autonomy concerns the degree to which action and experience are initiated and governed by "the self," in accord with self-endorsed values, needs, and intentions. Relatedness is the need to feel recognized and accepted by those who are viewed as significant.

Motivation studies in management typically discuss ways for managers to provide feedback, rewards, and a sense of belonging to employees in order to boost their internalization of organizational goals. However, most of these studies do not differentiate between extrinsic and intrinsic motivations, despite the fact that these concepts underlie many work-motivation theories. A recent review of motivation theory in the management literature calls for integration of various motivation sub-theories that have developed in the management field (Locke and Latham, 2004). As they show, intrinsic and extrinsic motivation concepts are interwoven into different theoretical concepts in the growing literature, but they highlight that empirical work which explicitly tackles "extrinsic" versus "intrinsic" motivation in organizational setting is still scarce. One exception is Baard, Deci, and Ryan (2004), who find that if employees possess a higher sense of autonomy (they feel relatively independent and in charge of their own work), their intrinsic motivation is higher, which increases their work performance. Similarly, Ilardi, Leone, Kasser, and Ryan (1993) use self-reported measures of intrinsic motivation to find that higher intrinsic motivation increased self-reported measures of job satisfaction in the workplace. However, by and large, there are almost no systematic econometric analyses that credibly separate and distinguish between intrinsic and

[8] See Gibbons (1998) for a good review.

extrinsic motivation, and even fewer that actually assess their relative importance in economic production and exchange.

The field of economics has historically not embraced the concept of intrinsic motivation, though in recent years it has begun to garner some attention. For example, a theoretical paper by Bénabou and Tirole (2003) provides conditions under which external rewards may undermine intrinsic motivations. Murdock (2002) models intrinsic motivation as generating some surplus in addition to economic returns of a trade. He assumes that people care about what they do (intrinsic motivation) in addition to being paid for it. He also assumes that people derive utility from accomplishing goals, and accomplishing goals is independent from any financial reward. Thus intrinsic motivation determines level of effort in the task. The net returns from a project consist of returns from intrinsic motivation and financial returns. The net returns could be still positive even if financial returns are negative. So, a firm can commit to a project in cases of negative financial return provided intrinsic motivation would increase employees' participation effort and thus net return would be positive. The motivating example for this model is Merck, Inc.'s development of a drug that cures river blindness. The financial returns of the project was negative ex-ante, but the company decided to complete the project because not going through would have had negative impact on researchers' morale.[9] Kreps (1997) discusses the challenge in trying to understand the interplay between intrinsic and extrinsic motivation, especially when the extrinsic rewards are not sharply defined. However, his overall assessment is that understanding this interaction is important, though it may result in "messy" and partially non-economic considerations involving the form of individuals' utility functions.

The Open Source Software domain is a good setting for this type of research, as it depends on both intrinsic and extrinsic factors to function. Lakhani and Wolf (2005) operationalize motivation according to the social psychology constructs of Deci (1971), using survey instruments on a sample of developers listed in SourceForge.net. They also separated intrinsic motivation into enjoyment-based intrinsic motivation and obligation/community-based intrinsic motivation (in accord with Lindenberg (2001), who called for studies on norm-based intrinsic motivation). The survey items to measure intrinsic motivation were based on self-reported responses to various reasons for why developers contribute, such as whether doing so would improve their programming skills, whether the contributions were made for results that might be useful for their own work, and whether they contributed because doing so would enhance their professional status. They found that intrinsic motivation is in fact the strongest driver of OSS project participation.

Our paper closely relates to Roberts, Hann, and Slaughter (2006), who distinguish motivations as pure extrinsic (for pay), internalized extrinsic (voluntary but for personal use benefit or status), and pure intrinsic (solely for the joy of doing it). Their empirical setting, like ours, is the OSS community. They

[9] In a related vein, Akerlof and Kranton (2003), Bénabou and Tirole (2003), and Francois (2000) also discuss the concept and implication of nonpecuniary benefits.

also derive their hypotheses from self-determination theory of motivation and then determine relationships between motivation, participation in OSS projects, and project performance. However, in terms of the empirical methodology, our study differs substantially from theirs, which relied on 288 responses to an online survey. Aside from being substantially larger, our sample also eliminates any bias that may result from the self-reported nature of online surveys.

We identify three types of developers based on the expected relative importance they attach to intrinsic motivation when making code contributions: *open*, *closed*, and *mixed*. For this, we assume that developers reveal themselves as "motivated agents" – which is one form of intrinsic motivation – if they are members only of projects that have open licenses, because these projects adhere more closely to the original philosophy of OSS. Such attachment to projects with open licenses – and disinclination to contribute those with more closed licenses – indicates an identification by such developers with the (OSS) objectives of such projects. This is how Besley and Ghatak (2005) define "motivated agents" – alignment of preferences between the employee and employer.

Conversely, we assume that developers reveal a preference for extrinsic motivation if they belong only to projects with closed license types. We also argue that anonymous developers are likely to resemble the intrinsic developer type, because it is doubtful that these developers could benefit from external benefits such as reputation or peer recognition. We consider as *mixed* those developers for which there is no clear evidence on whether they should care more about intrinsic or extrinsic motivation. Put simply, our empirical investigation 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 proceed by identifying four classes of motivations which run the spectrum from highly intrinsic to highly extrinsic: *Pure intrinsic motivation, Utility/Learning, Reciprocity,* and *Reputation.* The literature and available small-sample survey evidence suggest a role for each of these motivations in open source development (e.g., Haruvy et al., 2003; Lakhani and von Hippel, 2003; Hertel, Krishnan, and Slaughter, 2003; Lerner and Tirole, 2002). Out of the four motivation types, we can see that reputation relies most directly on extrinsic mechanisms. This is because the very notion of "reputation" requires an external agent to form an opinion of the subject. At the other extreme, pure intrinsic reputation is simply defined as lacking extrinsic dimensions. Thus, these two are at the tail ends of the motivation spectrum. On the other hand, reciprocity and utility/learning are likely to lie along the continuum, since both share features of internal and external motivations. The hypotheses we formulate aim to assess the relative importance of each motivation type in driving OSS contribution, while acknowledging the important point that an agent 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.

Pure intrinsic motivation. As discussed earlier, a salient feature of OSS is that many of its developers

have a strong ideological preference for keeping their source code fully open. In economic terms, this means that for a developer who is affiliated only with open projects, the utility derived from making a code contribution is larger if it is made to a project that is itself open. In the extreme, this utility would be zero if the contribution 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. This 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 because these more closely match the OSS philosophy.

The debate between advocates of open source and proprietary software has been polarized, with strident criticism from both sides of the divide (for discussion, see Lerner and Schankerman, 2010). In this context, there may also be developers ideologically motivated against contributing to HO projects, whose licenses may be viewed as "anti-property rights." [10] To the extent this view is widely held, the ideological motivation may also induce sorting by closed developers. It is important to emphasize that this type of sorting should be considered a form of intrinsic motivation, because it is ideologically based.

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; Markus et al., 2000). Here, the benefits are much more starkly extrinsic, since it is clear to see how status and respect from the other members of a community may motivate contributors to expend effort even in the absence of pecuniary rewards. However, their relative 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

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

(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 four 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.

The second type of reputation gain is basic peer recognition, which is unrelated to labor market payoffs. Peer recognition should also be greater for larger projects, projects that reveal patch outcomes, and those aimed at programming tools rather than end users. Whether peer recognition should be regarded as intrinsic or extrinsic motivation is less clear. On the one hand, peer recognition is likely to depend on the quality of a developer's contribution and to thus also lead to more externalized rewards such as status or monetary compensation. On the other, peer recognition by itself is consistent with the relatedness element of the SDC theory – the need to feel recognized by those who are viewed as significant. Nonetheless, it is important to note that whether reputation works via the labor market or peer recognition, it should play no role for anonymous contributors, since they cannot benefit from it. Moreover, among non-anonymous developers, we expect that reputation would be less important for contributors who sort (primarily) into open projects. This is because intrinsic motivation (in the form of motivated agents) plays a more important role for them. In 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 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 the outcome of the contribution. Anonymous developers should either be unaffected by this, or be negatively affected, if this 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 that are sponsored by corporations. However, anonymous and open developers should either be unaffected by this, or be negatively affected, if this crowds out their intrinsic (ideology) motivation.

Finally, the 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 peer recognition. Therefore, to the extent that reputation is valuable to extrinsically motivated developers, we expect these gains to be larger when the intended audience for the receiving project is developers rather than end users. This leads to:

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.

Reciprocity. The earliest proponents of open source emphasized the role that reciprocity (sometimes also known as "gift culture") plays in sustaining incentives for innovation. The hypothesis is that developers who are members of a project may make contributions either in direct response to, or anticipation of, contributions made by other developers to their project. Evidence from surveys of programmers certainly points to its importance. The idea that reciprocity can be sustained as an equilibrium over time is also well grounded in the economic theory of repeated games, though whether its assumptions fit the open source context is open to dispute. This equilibrium can be rationalized by the theory of repeated, non-cooperative games if players (code contributors) can detect and effectively punish those who deviate from reciprocity strategies. However, within the open source context, it is difficult in practice to identify such deviation (i.e., how can one know what level of contributions reveals such deviation) and to punish it. It may also be easier to sustain such reciprocity in particular kinds of projects, where the set of potential contributors is more readily known and thus non-reciprocal behavior is more easily spotted.

This construct may impact both intrinsic and extrinsic motivation. If the OSS developer feels that doing work now may result in future benefits through others' reciprocity, then this will elicit an extrinsic motivation. On the other hand, if the developer feels compelled to contribute in order to "give back," then reciprocity may elicit an intrinsic motivation. Empirically, there is some evidence that both effects will be relevant. For example, Shah (2006) finds that one of the most important reasons for developers to contribute to open platforms was a sense of reciprocity. Apart from deriving satisfaction from developing a code, contributors felt that helping others in the community was important because they had benefited from others' contributions.

The psychological contract theory also suggests that some contracts may involve strong emotional ties and loyalties, which necessitate reciprocal appreciation of intrinsic motivation (Rousseau and McLean Parks, 1993; Schein, 1965). This goes beyond simple transactions and involves socio-emotional implicit agreements or perceptions of agreements. From this perspective, the reciprocity encountered in organizations may have both intrinsic and extrinsic components. More relatedly, current work exploring the motivations behind OSS has pointed to reciprocity as a plausible mechanism. Lakhani and von Hippel (2003) find that reciprocity is the most important reason contributors cite for posting answers on Usenet

groups. These people either repay for the benefits they had received or contribute in expectation of benefitting from the community in the future. But it is important to note that we are agnostic as to whether reciprocity works through extrinsic or intrinsic channels (or both). This leads to:

Hypothesis 3 (reciprocity): 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.

Utility/learning: Like reciprocity, utility and learning motivations can consist of intrinsic and extrinsic components. Developers may contribute code to projects because they enjoy doing so (intrinsic) or because they learn from the process, which can provide benefits later (extrinsic). In addition, they may hope to influence the direction of the software project in ways from which they expect to benefit later. Such learning and influencing benefits from making contributions are likely to be stronger if the contributing and receiving projects use the same programming language or operating system.[11] Thus:

Hypothesis 4 (utility/learning): Projects are likely to receive more contributions from developers who are affiliated with other projects that have the same programming language or operating system.

3. Data

The data are taken from SourceForge, the largest web host for open source software projects. SourceForge provides a publicly accessible platform, introduced in 1999, where 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 code contributions. The final data set covers the 211,705 open source projects registered on SourceForge over this period, and all code contributions made to these projects. While our raw sample includes information on all 211,705 projects registered on SourceForge.net as of November 2010, the distribution of code contributions by receiving projects is highly concentrated, with only 6,316 (3 percent) receiving any external contributions over the ten years covered by the data. Interestingly, including internal contributions only raises that figure to 3.4 percent. The majority of the analysis focuses on the smaller sample of active projects. The Data Appendix provides a detailed description of the data collection, as well as a number of consistency checks.

We begin by defining some key terms. A *contribution* is defined as any contributed code that aims to advance the software in a particular manner (whether or not it is accepted by the project manager). There is no limit (minimum or maximum) on the number of lines of code it involves. This definition is

[11] The similarity in language may also be a driver because developers incur effort costs in making code contributions, and they may be more likely to contribute when these costs are lower. This could be the case when the programming languages of contributing and receiving project are similar. We acknowledge that it is not possible to distinguish empirically between the role of lower effort costs and utility/learning benefits.

maintained throughout the sample period. Our definition is standard in the empirical literature on code contributions using SourceForge data. We exclude software bug reports and fixes, which are more minor interventions.

The *contributing project* is the project with which the developer making the patch contribution is affiliated as a registered member. If the developer belongs to more than one project, we treat each separate project as a contributing project. We treat contributions made by anonymous developers, and by developers are not registered as members of any projects, as separate categories. We define the *receiving project* as the one to which the patch contribution is submitted. The distinction between the contributing and receiving projects is central to our empirical analysis of sorting, because our objective is to establish whether developers affiliated with certain types of projects (in terms of their license type) systematically target certain kinds of projects. Finally, we define *external contributions* as patches submitted by developers who are not registered members of the receiving project, and *internal contributions* as patches submitted by developers to projects of which they are members.

The key variables in the empirical analysis are as follows:

Project License Type: The most important project characteristic we consider in this paper 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 code and any of its modifications can be subsequently embodied in commercial software and (2) whether modifications to the code have to remain open source (i.e., the binary code must remain open and accessible).[12] 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 GPL (General Purpose License) license. It requires that any file, regardless of code origin, which is combined under certain circumstances with a file under GPL 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 source, but it can be used commercially. There are no restrictions on the

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

license conditions of the modifications and extensions of the code, provided that they remain 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.

Projects may have more than one license type. In cases of multiple licenses (7 percent of the sample projects), we classify the project as HO if at least one of its licenses is highly open, and as C if all of its licenses are closed.[13] The remaining projects are classified as O. In the complete sample, 47 percent operate under an HO license and 43 percent under a C license.[14] Among projects that receive at least one code contribution, 60 percent operate under HO licenses, and 25 percent under C licenses. This difference reflects that fact that highly open projects receive fewer contributions per project than those under closed licenses.

As mentioned earlier, a project can receive contributions from its members (internal contributions) or developers who have no formal affiliation with the focal project (external contributions). In the sample, 39 percent of contributions are internal. The remaining 61 percent are external, coming either 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). In what follows, we focus the analysis of sorting behavior on the pattern of these external contributions. In the empirical section we study the impact of different types of contributions, both internal and external, on the performance of the receiving projects.

Developer Types: We assign each developer a "type" based on the types of projects to which she belongs as a registered member. We classify a developer as *open* if all of the projects to which she belongs operate under highly open license, 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*. The majority of developers belong to very few projects – the mean number of projects of which a developer is a member is 1.4 (median is 1, 99^{th} percentile is 7). In addition, we use two other categories: "Anonymous" developers are those who submit contributions without revealing their identity to the receiving project manager or members. "Non-members" are contributors who are not registered as members of any project.

Of the 149,956 developers registered on SourceForge (i.e., having a unique user name), 68 percent have no affiliation to any project, 15 percent are open developers, 9 percent are closed developers, and

[13] 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.

[14] In their study of the determinants of project license type, Lerner and Tirole (2002) use an earlier and smaller sample but find a very similar distribution of license types.

the remaining are classified as mixed. Of the total sample of developers, only 22,512 (15 percent) make at least one contribution to projects with which they are not affiliated ("external contributions") over the ten-year 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.[15] Size is a central measure in our analysis because it is our key test of Hypothesis 2a on the reputation motive.

Most projects are small – the median project has one member (mean is 4.1). The distribution is sharply skewed, however – the project at the 90th percentile of the size distribution has 10 members (99th percentile is 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.

Resolution of the code contribution: From information on SourceForge, we know whether the open source project reveals to outsiders whether an individual code contribution has been accepted (i.e., incorporated in the project software). In total, 46 percent of projects registered on SourceForge reveal the outcome of (at least some) contributions.

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.[16] About 30 percent of projects receiving contributions are developer-oriented and 18 percent target end users. We also include a separate category for projects that do not specify intended audience (16 percent of projects).

Programming Language: 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 (Data Appendix for details). We also include a separate category for projects that do not specify their programming languages (14 percent of projects).

Operating System: 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 (Data Appendix for details). We also include a separate category for projects that do not specify their operating system platform (20 percent of projects).

[15] An alternative measure of size is the total number of patches received by the project. However, because we want to explain the pattern of developer code contributions, it would be problematic to treat this measure as exogenous for small projects since the developer's decision to contribute would affect the measure of project size.

[16] The End User category includes end users/desktop and advanced end users. The "Other" category, which account for about four percent of projects, includes mainly aerospace, education, science/research and healthcare.

4. Econometric Specification

Our primary objective is to estimate the effect of project characteristics on the pattern of code contributions. To do this, we first aggregate patches into cells, where each cell is defined by a set of characteristics of the contributing 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 contributing and receiving projects defining those cells.

Since the number of contributions is an integer, we use an econometric model for count data. We adopt a Negative Binomial specification

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

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 that 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.[17]

Our primary interest 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. For the contributing project, we focus on the developer type. As explained earlier, we infer the developer type from the project affiliations (membership) of the developer. Using the contributing developer type allows us to examine whether developers sort – i.e., 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 on SourceForge, and this will depend on how long the project has been registered. Each of these dimensions of the contributing and receiving projects are defined as dummy variables. An example of a cell is the total number of contributions (over the sample period 2001–2010) made by highly open developers 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.[18]

[17] 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 $\phi = 0$.

[18] Some cells have only dormant projects which receive no contributions at all over the sample period, in which case we drop them in the estimation procedure.

The key point to recognize here is that the dependent variable, $Y_{c,r}$, is defined on the basis of a set of contributing and receiving project characteristics at the cell level, not at the level of the individual contribution. These cells are defined in terms of the set of developer type (the contributing project characteristic we focus on) and the receiving project characteristics described above. Thus there is no ambiguity about the contributing developer type in the sorting, even though we analyze flows of contributions at the cell level.

As explanatory variables in the regressions, we include a complete set of dummy variables for the contributing developer type and the receiving project characteristics, and their interactions. 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 same cell since that will affect the flow of contributions.[19] We include projects that do not receive any contributions over the sample period provided that they are in a cell where at least project received one or more contributions. We drop cells if all projects in the cell received no contributions over the entire sample period.

We must normalize one of the coefficients on the interaction dummy variables between developer and license type (since we also control for additive effects in these dimensions). 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, i.e., relative to the expected number of contributions by open developers to projects with HO licenses.

The identification assumption we use is that the license type, and other project characteristics, are exogenous with respect to the *individual developer's* decision to contribute. The main source of concern is unobserved project quality, which might be correlated both with 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 – not on the level effects given by (α, β). Unobserved heterogeneity will induce bias only if it is correlated with these interactions. While higher-quality projects may attract more contributions, it is not clear why this should systematically affect one type of contributor more than the other.

There is also a concern that our measure of project size – the number of registered members – might be endogenous if unobserved project quality both attracts more members and makes it more likely that developers will contribute patches to the project. While we control for many observed project

[19] We do not control for the number of potentially contributing developers because there is no way to do this for one important category, 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 the sorting behavior of each developer type separately.

characteristics that might be correlated with project quality, we cannot rule out the possibility of an unobserved element to quality. If this is present, we expect the coefficient on project size to be biased upward. However, and this is the key point for our analysis, there is no obvious reason to expect this upward bias to be different for different types of developers. As we will show, we find that project size matters much more for closed developers – where the theory suggests that labor market signaling should be more important. Thus, while we cannot rule out the possibility that our estimated marginal effect of project size on contributions may be upward biased, we believe our key inference is robust to this concern.

Finally, one might be concerned that project size itself depends on the project manager's initial choice of open source license and other project characteristics. What would be the consequences for our findings if this is so? Suppose that the project license type (or other characteristics) affects not only contributions but also membership. In the econometric work on sorting, we exclude internal contributions. It is reasonable to assume that developers who become members of a project are those most interested in the project, and thus most likely to contribute heavily to the project if they did not become members. This implies that we would *underestimate* the impact of sorting on the actual flow contributions the project receives, since we do not capture the internal contributions made by developers who are induced to register as members by the manager's initial choice of project characteristics. In this sense, our empirical conclusion that there is strong sorting by developers is conservative.

5. Descriptive Statistics

Developers in our sample make 103,712 external code contributions. Table 1 shows how these contributions distribute across developer types. Of the total, 31 percent come from non-members, 18 percent from anonymous contributors, 14 percent from open developers, and 13 percent 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. However, while they contribute fewer code contributions, in Section 7 we show that the contributions by non-members have a much larger impact on project performance than the other categories of external contributors.

Open developers tend to be members of fewer projects than closed and mixed developers (means of 1.5, 2.0, and 2.9 projects, and 99^{th} percentile values of 5, 11, and 12, respectively). However, there is no substantial difference in the number of distinct projects to which developers of different types contribute (not reported in the Table 1): mean values are 1.4, 1.5, and 1.5, respectively. Non-member developers tend to focus their contributions more narrowly (an average of only 1.1. projects) than developers who belong to projects.

Table 2 summarizes key aspects of sorting behavior by developers: how they target their contributions

toward different types of projects. Several interesting patterns emerge. First, there is strong sorting of contributions on project license type. Both anonymous and open developers are much more likely to contribute to projects operating under highly open licenses. For anonymous developers, more than 80 percent of their contributions go to such projects, with a further 10 percent 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 percent that go to moderately open projects. In sharp contrast, closed developers focus more heavily on projects with closed licenses – 36 percent going to such projects (as compared 9 and 16 percent for anonymous and open developers).

Second, there is strong sorting on the intended audience of the project. 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. In sharp contrast, anonymous developers are almost four times more likely to contribute to projects aimed at end users compared to projects targeting other developers. Open developers do not seem to favor either end-user or developer projects. While there does not appear to be any clear sorting by open developers in this table, once we control for other factors in the econometric analysis (Section 6) we find that there is also strong sorting toward end-user projects by open developers as well. This pattern of sorting on intended audience of the software project is interesting because, given the importance of cumulative innovation in software, programming tools are likely to contribute more to the long-run technological advance in this sector than end-user products.

Third, closed developers sort much more strongly on the size of the project as compared to anonymous or open developers. For example, 43.9 percent of contributions by closed developers go to projects with more than 10 members, whereas the corresponding figures are 30.7 percent for open developers and only 17.7 percent for anonymous contributors.

These two findings – sorting by intended audience and project size – are consistent with the hypothesis that closed developers are more driven by the motive to build a reputation and signal it in labor markets, since programming skills are more effectively signaled via contributions to developer tools (where the audience has the technical skills to evaluate them), and to more visible projects. In the econometric analysis, we will also provide further evidence that supports the importance of the reputation motive, while we also consider and try to rule out other explanations for this observed sorting behavior.

6. Econometric Results

6.1. Baseline Specification

Table 3 presents the estimated parameters (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. The regression also includes a complete set of additive dummy variables

for the receiving project characteristics (parameters omitted for brevity). 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 developer to a project with an HO license (this is 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.[20] [21]

The empirical results show that there is strong sorting behavior by developers. Turning first to column 2, we see that 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. Since the average number of contributions by open developers at the cell level is 9.3, this is equivalent to a 17.4 percent reduction. Moving from an open to a closed license is associated with an *additional* reduction in contributions of 0.34, or 3.7 percent. The χ^2 test strongly rejects the hypothesis that there is no sorting by license type for highly open developers (p-value$<$ .001).

Contributions by anonymous developers cannot be driven by career concerns because they do not reveal their identity and cannot gain from peer recognition or labor market signaling. Thus anonymous contribution activity indicates either the importance of ideology as motivation ("motivated agents") 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 in Table 4 shows that anonymous developers sort in a way similar to highly open developers. The sorting coefficient on HO licenses is not statistically different from zero, which is the same as for open developers. Moreover, anonymous developers show the same disinclination to contribute to projects with less open or closed licenses. This is shown by the statistically significant, negative sorting on open and closed licenses (-2.20 and -2.95, respectively). 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). They 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 an C to an HO license reduces the number of contributions by closed developers by 1.53. This represents an 18.7 percent fall

[20] Differences in the sorting coefficients across developer types, for a given license, reflect differences in both the numbers of potentially contributing developers of each type and their intensity of contributions.

[21] Table 3 includes the mean number of contributions by different developer types at the "cell" level, which we use to compute percentage impacts in the discussion which follows. For completeness, we provide here more information on these differences. The mean, median and 99th percentile of the distribution of contributions at the cell level are as follows: Anonymous developers – 12.1, 1, and 114; Open developers – 9.3, 1, and 142; Mixed developers – 16.7, 9, and 180; Closed developers – 8.2, 0, and 145; and Non-members – 21.0, 3, and 326.

in their contributions at the cell level (= 1.53/8.2). Again, we decisively reject the null hypothesis that there is no sorting by closed developers (p-value <.001). 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 (multiple) 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 on membership and in their contribution activity.

Finally, the results in column 5 show that non-member developers exhibit behavior which is very similar to open and anonymous developers. In particular, non-member contributors sort toward highly open licenses. Moving from an HO to a C license reduces the number of contributions by non-member by 2.43 (= 2.65-5.08). Their average number of contributions at the cell level is 21.0, so this change in license type reduces their contributions by 11.6 percent.

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. If this is so, larger projects would attract more contributions from all developer types. Second, project size may be associated with more exposure and thus larger reputation gains.[22] But these gains can come both in the form of greater peer recognition and/or labor market signaling benefits. Thus they 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.

Nonetheless, we can distinguish between the project quality and reputation effects associated with project size in the following way. Reputation benefits should not be relevant to developers who contribute anonymously. In deciding where to contribute, however, anonymous developers are likely to prefer higher quality projects (e.g., if they get greater utility value from such projects). Therefore, we can infer the impact of project quality on contributions from the behavior of the anonymous type. Under the assumption 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.

Turning to the result, 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 ten percent increase in project size raises contributions by 0.266, which is only 2.2 percent 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

[22] Larger projects may be more visible, but an individual's contribution in a large project may be less salient. Thus it is not clear whether larger projects generate greater indivisual reputation gains. We are grateful to the Associate Editor for pointing this out. However, as we discuss in the text, the parameter estimates indicate that such gains are increasing in project size (controlling for pure utility value of contributing to larger projects).

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.[23] This finding strongly supports Hypotheses 2a on reputation and project size.

Finally, the test statistics in Table 4 confirm that intended audience, programming language, and operating system all significantly affect the pattern of contributions. We decisively reject the null hypothesis that each of these (sets of) dummy variables do not affect contributions (p-value $< .001$ for each of the three cases).

To summarize, our key empirical finding is that project characteristics, including license type, affect the level of contributions by each developer type. This means that the choices managers of open source projects make in this regard are important. We illustrate this point in more detail in Section 6.5.

To test our Hypothesis 2d, on reputation and intended audience, Table 4 explores sorting patterns on intended audience.[24] We add interaction terms for each developer type with their intended audience dummies: developer tools and end users. The pattern of results shows that there is strong sorting on intended audience both by 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 in the number of contributions of 2.12 (= 0.65+1.47). This change in intended audience type reduces their contributions by 22.8 percent (relative to the cell average number of contributions). Anonymous developers show a similar pattern. Moving from developers tools to end users raises their number of contributions by 1.25 (= -1.56+2.81), or 10.3 percent of average cell number of contributions. In sharp contrast, closed developers seem to express a strong preference for developer tools 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 percent 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.

These findings provide only partial support for Hypothesis 2d. We find the predicted sorting by closed developers 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 our

[23] We also estimated the baseline specification in Table 3 separately for different receiving project sizes. The results show the same pattern of sorting between developer type and receiving project license, for different size categories.

[24] It is important to note that license type is highly correlated with intended audience. This makes identifying sorting by license type and intended audience more difficult. In our sample, 87 percent of end-user projects have a highly open license, as compared to only 31 for developer projects.

prediction was that open developers would favor developer tools and anonymous would not sort on this dimension.

6.2. Robustness Analysis

Matching on programming skills and software architecture

One concern is that the observed sorting behavior might be due, at least in part, to 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 perform the following test. We exclude anonymous and non-member developers, focusing only on member developers, and define cells on the basis of the programming language. We define a new dummy variable that receives the value of one for cells where the programming languages of the contributing and receiving projects are the same and re-estimate the baseline specification with this additional control variable. In cases where a contributing developer belongs to multiple projects, the contribution is counted separately for each project, so the matching dummies are defined at level of the contributing-receiving projects pair.

The sorting results continue to hold in this extended specification which controls for programming language match. As expected, the effect of programming language match is large and highly significant, with a marginal effect of 0.45 (standard error of 0.042), relative to a cell average number of contributions of 1.8. Moving from a HO to a C license reduces contributions by open developers by 12 percent, and increases it by 9 percent for closed developers. These estimates are very close to those where we exclude the programming language match dummy (13 and 12 percent).

We also repeat the above analysis matching on the operating systems of the contributing and receiving projects. Matching on operating systems, as expected, strongly affects contributions (a marginal effect of 0.47 with standard error of 0.09, as compared to the mean cell contribution of 2.21). The licensing sorting effects remain robust. Moving from a HO to a C license reduces open developer contributions by 16 percent, and raises closed developer contributions by 16 percent.

Project quality

There is no reason to expect that unobserved project quality is systematically correlated with the interaction between project license and developer type. Nonetheless, we perform an additional test to check robustness by introducing two direct measures of project quality. The first is the cumulative number of downloads of the project on SourceForge *prior* to the year of contribution. The second measure is the lag between the 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 a burst 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, in order to conduct these additional tests we need to redefine cells more finely, including the year of contribution as an additional dimension.

The main results on sorting continue to 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 – which is consistent with downloads being an indicator of project quality – but we do not find any statistically significant effect for the contributions lag. Importantly, the pattern and magnitude of sorting are robust to these new controls. For example, the estimates imply that moving from an HO to C license reduces average contributions by anonymous and open developers by 32 and 24 percent, and raises contributions by closed developers by 27 percent. These impacts are actually somewhat larger than the baseline effects, confirming that differences in project quality (at least as we measure it) are not driving the sorting results.

Variation over time and removing outliers. We perform two final robustness checks of our sorting effects. First, the role of intrinsic motivation may be changing over time. 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 (the same pattern of results holds when we split the sample by contribution year instead). We find that sorting on license type holds for both periods, but sorting is actually substantially stronger in the 2006–2010 period. For the pre-2006 period, moving from an HO to C license reduces contributions by 12 and 24 percent for anonymous and open developers, and raises contributions by 23 percent for closed developers. For the post-2005 period, moving from an HO to C license reduces contributions by 28 and 30 percent for anonymous and open developers, and raises contributions by 41 percent for closed developers.

Second, the distribution of contributions is highly skewed – a small number of projects receive a large fraction of contributions, and a small number of developers make a significant share of contributions. Because there may be unobserved heterogeneity 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.

6.3. Extensions

In this section we discuss two additional experiments to refine our inferences about the role of reputation.

Public resolution of contributions

The first extension exploits information on whether the receiving project announces whether the contribution made by a developer is accepted or rejected. After a developer makes a contribution, the

project manager (or members) decides whether or not to accept it and to make the decision public on SourceForge. The decision is made for each contribution separately, but projects differ in the degree to which they make these outcomes public.[25]

We use this information to identify the reputation incentive more sharply. There are two theoretical predictions, both embodied in Hypothesis 2b. The first is that the decision to contribute for anonymous developers 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 by them 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.

The second part of Hypothesis 2b is that developers who are motivated by either peer-based or commercial reputation should be more likely to contribute to projects with public resolution than other developers. Thus we can test whether reputation matters directly by examining whether contributions by non-anonymous developers are higher for such projects. However, since 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 for at least 50 percent of the contributions they receive over the sample period.[26] With this threshold, 71.1 percent of the projects are identified 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 5 presents the results. Our earlier findings about sorting on license type are similar to the baseline results in Table 4. Both anonymous and open developers sort strongly on projects with highly open licenses, while closed developers systematically target projects with less open licenses. There are two new findings here. First, the estimated coefficient on the public resolution dummy variable is positive and statistically significant for all developer groups, *except* anonymous developers. This is a direct confirmation of the hypothesis that reputation is a motivation for contributions, and confirms Hypothesis 2b.[27] Moreover, public resolution has the largest impact for open and closed developers. The estimated

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

[26] We also tried two alternative thresholds – 25 and 75 percent – 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.

[27] The only other likely explanation for why public resolution matters for non-anonymous developers is that the decision

coefficient of 4.01 for open developers implies that they contribute about 60 percent more contributions on average to projects with public resolution (= 4.01/6.7), while for closed developers the figure is 54 percent (= 3.15/5.8). For the other developer groups the implied increase in smaller, about 15 percent.

The second important finding is that the estimated coefficient for anonymous developers is negative, and marginally significant, with a p-value of 0.066. These developers do not value public resolution – which is consistent with the theoretical prediction since 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 indicates that 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.[28]

Corporate sponsorship

Increasingly, large firms have invested substantial financial and technical resources in open source development, including paying employees to participate in such projects.[29] This is likely to include sponsorship and other form of involvement with projects registered on SourceForge. Knowing which projects have a substantial corporate involvement should help us pin down more sharply the role of labor market signaling, as distinct from reputation associated with peer recognition. The main prediction (Hypothesis 2c) is that developers motivated by commercial reputation – in particular, 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 since 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. 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 a anti-proprietary software ideology.

Unfortunately, SourceForge does not separately identify whether a project is corporate sponsored, or more generally the level of corporate involvement in any form. To examine this, we sent an e-mail survey of 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.[30] Therefore, to augment the usable sample, we performed extensive

to publish is a proxy for unobserved project quality. This cannot be ruled out, but in that case we would expect publishing to induce contributions from all develper groups, including anonymous developers. The empirical results are not consistent with this alternative explanation.

[28] The results in the text are based on a 50 percent threshold. Using a 25 percent threshold, the estimated coefficient (standard error) on the public resolution dummy for anonymous developers is -0.83 (0.64); with a 75 percent threshold, we get -2.14 (0.41).

[29] For discussion, see Lerner and Schankerman (2010). On IBM's involvement in open source, see http://www.research.ibm.com/journal/sj/442/capek.pdf

[30] The remaining responses classified projects as being initiated by individual developers, but it was not clear whether the

manual investigation of each remaining project to identify whether there was corporate involvement, either directly in the project or indirectly through offering a proprietary product that incorporates 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.[31]

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 percent of all contributions received in the complete sample. The distribution of contributions by developer types and corporate sponsorship (Table 6) shows clear sorting behavior. Anonymous developers target 86 percent of their contributions to not-for-profit projects, while 79 percent of contributions by closed developers go to corporate sponsored projects. Open developers favor not-for-profit projects (55 percent) but this sorting is weaker. This result indicates some role for signaling and other labor market considerations even for open developers.[32]

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 a set of dummy variables for developer type and receiving project characteristics.[33] Table 7 summarizes the results, and confirms the pattern found in Table 6. In column 1 we include only dummies for developer types. Closed developers are 32.4 percentage points (or 65.7 percent, evaluated at the sample mean) *more likely* to contribute to corporate projects than open developers. Anonymous developers are 36.2 percentage points (or 73.4 percent) *less likely* to engage with corporate projects than open developers. 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. There is no significant change in the point estimates. Interestingly, the estimates also show that corporate projects are least likely to use an HO license. Finally, in column 4 we add a full set of dummy controls for the intended audience, programming language, and operating system of the receiving project. Even with all these controls, we again 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 there is now no difference between anonymous and open developers. The results from this analysis strongly confirm

founding developers started the project with intent to commercialize or get corporate sponsorship in the future.

[31] Importantly, we did not classify projects as not-for profit as a default option. To be so classified, a project had to indcate clearly a non-commercial intent in their mission on their website or the various online forums we examined, and there had to be no company support (or stated intention to seek it), and no commercial products we could identify as building on the project.

[32] 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.

[33] We do not adopt the "cell" framework in this section because, despite the large number of contributions covered by the projects in this restricted sample, the "degrees-of-freedom" which are 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 break up by bundles of project characteristics in a meaningful way.

Hypothesis 2c.

6.4. 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. The primary hypothesis we are interested in is whether developers associated with highly open projects – those presumably most closely aligned with the open source ideology – are more likely to engage in reciprocity than other license types.

We focus on reciprocity at the project level (rather than individual contributing developer) because we want to be able to control for matching between projects on various dimensions. We define a contribution from a developer who is a member of project i to project j in year t 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 which previously contributed to it.

Table 8 presents some descriptive statistics on reciprocity. Three features stand out. First, reciprocity is rare at the project level. For the sample as a whole, only 4.9 percent of active projects (i.e., those that receive at least one contribution) are engaged in any reciprocal contributions.[34] However, these tend to be larger projects, which account for 37 percent of total patches received. Second, closed projects are more likely to involve reciprocity – 7.5 percent for closed versus 4 percent for highly open projects. This is also true when measured in terms of the percentage of contributions – closed projects with reciprocity account for 74.8 percent of received contributions, while it is only 20.6 percent for highly open projects. This finding is surprising since one might think that "intrinsically motivated agents," who care about open source ideology, would be more likely to reciprocate than commercially oriented developers. The fact that we find the reverse strongly suggests that reciprocity also has a signaling motive for the individual developer. Finally, while only a small minority of projects engage in reciprocity, it plays a large role for those that do. Among those projects, 44.6 percent of all contributions made by those projects are reciprocal, and again this is most pronounced for closed projects.

However, these two-way contributions do not necessarily reflect a reciprocity motive. They may occur because similarities between projects reduce the effort cost for developers to make such contributions. To pin down more sharply whether highly open projects are more likely to engage in reciprocity, we estimate Probit regressions of whether a contribution is reciprocal. To control for project similarity, we include a set of dummy variables that capture whether the contributing and receiving projects *match* on license

[34] While truncation may cause us to underestimate the occurrence of reciprocity somewhat – some reciprocal contributions may not have yet occurred within the period of observation – the number is so low that we think truncation is very unlikely to reverse this finding.

type, programming language, operating system, and intended audience. We also control for additive dummies for each of these dimensions.

Table 9 presents the results. Column 1 shows that reciprocity is more likely to occur 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, which is 64.2 percent of its sample mean. However, reciprocity is not affected by whether there is a match on programming language or operating system. This finding 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 contributing and receiving 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 there is no effect when projects are matched on more open license types. This again is inconsistent with the hypothesis that reciprocity is a major driver of contributions for developers strongly aligned with the open source ideology. While more research on this topic is needed, it appears from these results that reciprocity is more connected to signaling or other considerations that are, evidently, most important to developers associated with closed projects.

6.5. Application to the Management of Open Source Projects

In previous sections we showed there is sorting behavior among contributing developers. This implies that the choice of a project's open source license and other characteristics affects the flow of contributions to the project. For example, a more open license increases contributions by highly open developers but, at the same time, reduces those of other developer types. Thus project manager face a trade-off in selecting these characteristics.

Lerner and Tirole (2002) emphasize the trade-off that project managers face between the degree of proprietary control and the expected number of contributions. They argue that, while more open licenses weaken property rights over the software, such licenses presumably attract more widespread participation among potential contributing developers. Our evidence suggests that the impact of the choice of license is less clear-cut. More open licenses can either increase or reduce overall contributions – it depends on the sorting behavior of different types of developers. For this reason, the empirical facts about sorting are important for open source project managers.

We illustrate this point with two examples. The first shows how the openness of the license affects the total number of contributions received by a project. Let $\Delta_{l \longrightarrow L}$ denote the change in the number of contributions received that occurs when license type shifts from l to L, holding all other project characteristics constant. This will depend on how strong the sorting is between each type of developer and project license. We can use our estimated sorting coefficients from Table 4 to compute this effect.

Let λ_{dl} denote the sorting coefficient for developer type d and receiving project (cell) l. This gives 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 ($\lambda_{HO,HO} = 0$).[35] Then the effect of changing the license from type l to L on the total expected number of contributions, denoted by $\Delta_{l\to L}$, is given by $\Delta_{l\to L} = \sum_d(\lambda_{dL} - \lambda_{dl})$ where the summation is over the five different developer groups.

Using the parameter estimates from the baseline specification in Table 4, we get the following results: $\Delta_{HO\longrightarrow C} = -5.22$, $\Delta_{HO\longrightarrow O} = -6.80$, and $\Delta_{O\longrightarrow C} = 0.09$. To illustrate the interpretation, $\Delta_{HO\longrightarrow C} = -5.22$ means that switching from a highly open to a closed license, holding all other project characteristics constant, is associated with a decline of 5.22 (external) contributions. This represents a 9.2 percent reduction in the number of external patches received, computed at the cell mean. Switching from an highly open to an open license also reduces contributions, by about 12 percent.

These calculations indicate that, for the sample as a whole, the highly open licenses maximize the expected number of contributions. Of course, this result does not mean that managers should always choose highly open licenses, since these also involve constraints on the ability to appropriate commercial returns from the project. But the computation shows that the nature of the project license can make a real difference to the flow of contributions, and thus the rate of innovation in open source software. One should also take into account the possibility that sorting might also affect the quality of contributions – i.e., code contributions from different types of developers might have different marginal productivities In the next section, we study this question.

In the second example, we show that the choice of license should depend on the intended audience of the project. The reason is that the sorting behavior of different types of developers may itself vary with this (or other) project characteristics. To analyze this, we re-estimate the baseline model separately for projects whose intended audience is developer programing tools and end users, and use the estimated sorting coefficients to predict how the choice of license type affects total contributions for each project type. For brevity we do not report the full set of estimated parameters for each project type.[36]

For end-user projects, this computation yields the following results: $\Delta_{HO\longrightarrow C} = -5.16$, $\Delta_{HO\longrightarrow O} = -10.91$, and $\Delta_{O\longrightarrow C} = 5.77$. Computed at the cell mean number of contributions for end user projects, these figures imply that moving from a highly open to a closed license reduces external contributions by about 12.5 percent, and by about 26.4 percent if an open license were adopted. This indicates that the

[35] Recall that the sorting parameter captures the effect of the number of potentially contributing developers of that type, because we do not separately control for the number of developers in the regression.

[36] The split by intended audience substantially reduces sample size (number of distinct cells), and the estimates are less precise than for the pooled sample reported in Table 3. Nonetheless, the results again reveal sorting by developers on license type, but this varies by the intended audience and is less sharp than when we pool the data (this is because developers also sort on intended audience and this is controlled for in this exercise by splitting the sample of projects). In particular, open and anonymous developers tend to favor projects with HO licenses, especially in end-user projects, and closed developers favor closed licenses for developer tool projects.

highly open license maximizes the flow of contributions for end user projects. For developer tool projects, we get $\Delta_{HO \longrightarrow C} = 4.00$, $\Delta_{HO \longrightarrow O} = 8.28$, and $\Delta_{O \longrightarrow C} = -4.28$. These imply that the intermediate, open license maximizes contributions for developer tool projects.

How do these predictions line up with the actual choices project managers make? For end-user projects, the prediction is confirmed. In the sample, the highly open license is dominant for such projects: 87 percent have highly open licenses, with open and closed licenses accounting for only 5 and 8 percent, respectively. However, for developer tool projects, there is a more even distribution across license types than is predicted by our sorting coefficients: 31 percent for highly open, 33 percent for open, and 36 for closed licenses. Of course, project managers may also have other considerations in choosing a license, not just maximizing the inflow of contributions.

7. The Impact of Contributions on Project Performance

We have shown that software developers exhibit strong sorting behavior in their (external) contributions to open source projects. In particular, developers who are members of projects are more likely to contribute to other projects that match on the license type. As we show in the previous section, this sorting is important for potential managers of open source projects because it means that the selection of the type of open source license will affect the expected *volume and mix* of code contributions received by the project. In this section we go a step further by studying whether this matching behavior also affects the *quality* of the contributions received by the projects. To do this, we estimate the impact of different types of contributions on the performance of open source projects, and then examine whether the impact is greater when the contributing developer is matched (in terms of the license type) to the receiving project.

Beginning in 2006, SourceForge provides information that can be used to study the performance of projects. We focus on projects that received at least one contribution over the sample period 2001–2010. We were able to match about 66 percent 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. For this analysis, 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.[37]

We begin with a standard log-linear function that relates performance 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

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

[37] We also experimented with 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 measure is more likely to capture diffusion among software developers). The qualitative results with these alternative performance measures are similar to those reported in this section.

where η is a normally distributed error term where 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 patch 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

$$\ln Y_{it} = \alpha \ln \left(\Sigma_{j=1}^{J} \lambda_j S_{ijt}\right) + \beta Z_{it} + \eta_{it} \tag{7.1}$$

The assumption that code contributions from different sources are perfect substitutes is reasonable since they all represent the same type of input in the production (performance) function – new software code for the same program.[38] We need to normalize one of the λ parameters – we set $\lambda_1 = 1$,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 type one. The coefficient α is the elasticity of performance with respect to the aggregate stock of contributions.

In all of the performance regressions, we normalize the coefficient on the stock of contributions from non-member developers (the choice is immaterial and only affects the interpretation of the parameters). Each stock is computed as the cumulative number of contributions of the specified type from the project's inception (registration date on SourceForge). In all regressions we include controls for the intended audience, programming language, operating system, size of the project (number of members), and the year of registration on SourceForge. We estimate this regression by nonlinear least squares, and report standard errors clustered at the project level.

Table 9 reports 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 (including non-members). We find that project performance is strongly related to contributions received, with a significant elasticity of 0.307. This result underlines the importance of choices made by project managers that influence the flow of contributions, including license type and other project characteristics. In addition, performance is positively related to the project size, and also varies across the other project characteristics used as control variables (we strongly reject the hypothesis that these characteristics do not affect performance, p-value$<$.001).

However, this association between the stock of contributions and downloads hides an important difference. In column 2 we break down contributions into three separate stocks: *internal contributions* (from

[38] This modified Cobb-Douglas production function specification was first used by Griliches (1986), who used it to study the impact of basic and applied research on productivity.

developers who are members of the same project), *external contributions* (from developers who do not belong to the project), and *non-member contributions* made by developers who do not belong to any SourceForge project. The result is striking – while the overall elasticity is virtually unchanged, at 0.304, 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 marginal productivity for internal contributions is 0.344, only a third as large as for non-member developers, while the marginal productivity for external contributions is 0.620. Despite the difference in point estimates, we cannot reject the hypothesis that internal and external patches have the same marginal productivity (p-value =0.08), but we strongly reject that they are the same as for non-member developers.

Finally, in column 3 we disaggregate the external contributions into two groups: *Matched*, where the license type of the contributing developer is the same as the receiving project, and *Unmatched*, where the license type is different. The results show that Matched and Unmatched contributions have very similar (not statistically different) marginal productivities, estimated at 0.174 and 0.169, respectively. Moreover, these are also very similar to the marginal productivity for internal contributions, estimated at 0.222. However, all three of these are dramatically smaller than the marginal productivity of contributions made by non-member developers (normalized to unity).[39]

This analysis of project performance reveals two key facts. First, contributions do matter for project performance. Second, while sorting behavior of developers affects the flow of contributions to different types of projects (as we showed in Section 6.1), this sorting does not affect the marginal productivity of the contributions. The one exception to this is contributions made by developers who do not belong to any project, which have much larger impact on performance. One possible explanation for this difference is that the incentives for signaling may be greater for developers who do not belong to any projects and that, as a consequence, they make more effort in providing higher quality contributions. Unfortunately, we cannot test this interesting hypothesis in the absence of a direct measure of the quality of code contributions.

8. Concluding Remarks

This paper explores the role of intrinsic motivation, reputation, and reciprocity motives in driving open source software innovation. The empirical analysis exploits a large-scale dataset with detailed information about code contributed to open source software projects, as well on and the characteristics of contributing and receiving projects. We study the pattern of contributions by five distinct developer groups – open,

[39] In these regressions we include contributions made by anonymous developers together with highly restrictive developers, since we found in Section 6.1 that their sorting behavior was very similar. The results are robust, however, to including a separate stock of contributions for anonymous developers. As an additional robustness check to account for the skewness of project downloads, we windsorized the data at both the 97th and 99th percentile and re-estimated all specifications in Table 9. The results are very similar to those reported in the table.

closed, mixed, anonymous and non-member – and infer their underlying motivations from the "revealed preference" of projects chosen.

The central finding is that developers of different types strongly sort on observed project characteristics – most notably, the openness of the license, project size, and corporate sponsorship. The empirical pattern of sorting behavior points to an important role for (intrinsically) motivated agents, as well as reputation, especially in terms of commercial reputation for closed developers. To a lesser extent we find support for the reciprocity motive in sustaining open source software innovation. Finally, we show that code contributions affect the performance (quality) of open source projects. However, contributions by non-members seem to be a much more important determinant of quality than those made by project members.

There are two main directions for further research. The first is to develop and estimate an empirical framework that incorporates both the choice of license contract and the resulting sorting effects – for example, by integrating the work in this paper with the model of Lerner and Tirole (2002). A second direction is to study in more detail how contributions from different types of developers, both members and non-members, affect the performance of open source projects, and to integrate this inquiry with knowledge spillovers among open source contributors.

A. Appendix

A.1. Data construction

We developed specialized web crawler software that extracts information from the SourceForge website (a procedure that takes about a week to complete). We used two different versions of the crawler software to implement the extraction (changes in the website format required this): November 2005 and September 2010. In each extraction we retrieved information for all projects listed on SourceForge, as well as all relevant information for each project. This repetition of the extraction allows us to check the consistency of information both within and across projects over time. This is important in order to address the concerns, raised by Howison and Crowston (2004), about arbitrary dumping of information on SourceForge.

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, only 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 has been a shift in the distribution of projects across license types from 2005 to 2010, with closed licenses more heavily represented in the later sample. The composition of the 2010 sample is as follows: 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 over time toward projects with closed licenses is likely to reflect the increasing involvement of corporate-sponsored open source activity.

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 presumably the more serious concern as it associated with incompleteness of the history of the object of interest. The attrition may vary over time and thus can potentially bias our results (for example, 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 the complete historical information on submission of 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 an active project 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 – i.e., 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 a very accurate information on the history of code submission. We conclude that there is no evidence of substantial 'dumping' or 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 from SourceForge to other websites. For example, 55% of the dropped contributions belong to *Python*, which moved to the website http://www.python.org/. However, during the period that *Python* appears on the SourceForge website, 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 registered on SourceForge change over time. We turn first to our most important characteristic – 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

We use five categories in the empirical analysis. The programming languages included 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

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

References

[1] Akerlof, G., and Kranton, R. (2003). Identity and the economics of organizations. *Journal of Economic Perspectives*, 19(1): 9-32

[2] Ackerberg, D., and Botticini, M. (2002). Endogenous matching and the empirical determinants of contract form. *Journal of Political Economy,* 110(3): 564-91

[3] Baard, P., Deci, E., and Ryan, R. (2004). Intrinsic need satisfaction: a motivational basis of performance and well-being in two work settings. *Journal of Applied Social Psychology,* 34(10): 2045-2068

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

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

[6] Bruns, B. (2001). Open sourcing nanotechnology research and development: issues and opportunities. *Nanotechnology* (Institute of Physics), 12: 198-210

[7] Deci, E. (1971). Effects of externally mediated rewards on intrinsic motivation. *Journal of Personality and Social Psychology,*18: 105-115

[8] Deci, E. (1972). Intrinsic motivation, extrinsic reinforcement, and inequality. *Journal of Personality and Social Psychology,* 22: 113-120

[9] Deci, E., and Ryan, R. (1985). *Intrinsic Motivation and Self-determination in Human Behavior.* New York: Plenum Press.

[10] Deci, E., and Ryan, R. (2000). Intrinsic and extrinsic motivations: classic definitions and new directions. *Contemporary Educational Psychology* 25: 54–67

[11] Delfgaauw, J., and Dur, R. (2007). Signalling and screening of workers' motivation. *Journal of Economic Behavior and Organization*, 62(4): 605-24

[12] Delfgaauw, J., and Dur, R. (2008). Incentives and workers' motivation in the public sector. *Economic Journal*, 118: 171-91

[13] Francois, P. (2000). Public service motivation as an argument for government provision. *Journal of Public Economics,* 78(3): 275-99

[14] Frey, B. (1997). *Not Just for the Money.* Cheltenham: Elgar Publishers.

[15] Frey, B., and Oberholzer-Gee, F. (1997). The cost of price incentives: an empirical analysis of motivation crowding-out. *American Economic Review*, 87(4): 746-55

[16] Gibbons, R. (1998). Incentives in organizations. *Journal of Economic Perspectives* 12(4): 115-132

[17] Glazer, A. (2004). Motivating devoted workers. *International Journal of Industrial Organization,* 22(3): 427-40

[18] Griliches, Z. (1986). Productivity, R&D, and basic research at the firm level in the 1970s. *American Economic Review,* 76(1): 141-154

[19] Hann, I-H., Roberts, J., Slaughter, S., and Fielding, R. (2004). An empirical analysis of economic returns to open source participation. Working paper 2006-ES, Tepper School of Business, Carnegie-Mellon University.

[20] 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.

[21] Hertel, G., M. Krishnan 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

[22] Hope, J. (2008). *Biobazaar: The Open Source Revolution and Biotechnology.* Cambridge, MA: Harvard University Press.

[23] Howison, J., and Crowston, K. (2004). The perils and pitfalls of mining SourceForge. In *Proceedings of the International Workshop on Mining Software Repositories:* 7-11

[24] Ilardi, B., Leone, D., Kasser, T., and Ryan, R. (1993). Employee and supervisor ratings of motivation: main effects and discrepancies associated with job satisfaction and adjustment in a factory setting. *Journal of Applied Social Psychology* 23(21): 1789-1805

[25] Johnson, J. (2002). Open source software: private provision of a public good. *Journal of Economics and Management Strategy*, 11: 637-62

[26] Johnson, J. (2004). Collaboration, peer review and open source software. Unpublished working paper, Cornell University.

[27] Kreps, D. (1997). Intrinsic motivation and extrinsic incentives. *American Economic Review, Papers and Proceedings*, 87(2): 359-364

[28] 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.

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

[30] Lerner, J., and Schankerman, M. (2010). *The Comingled Code: Open Source and Economic Development.* Cambridge: MIT Press.

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

[32] Lerner, J., and Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics,* 52: 197-234

[33] Lerner, J., and Tirole, J. (2005). The economics of technology sharing: open source and beyond. *Journal of Economic Perspectives,* 19(2): 99-120

[34] Lindenberg, S. (2001). Intrinsic motivation in a new light. *Kyklos,* 54(2/3): 317-342

[35] Locke, E., and Latham, G. (2004). What should we do about motivation theory? Six recommendations for the twenty-first century. *Academy of Management Review* 29(3): 388-403

[36] Maurer, S., and Scotchmer, S. (2006). Open source software: the new intellectual property paradigm. NBER Working Paper 12148

[37] Murdock, K. (2002). Intrinsic motivation and optimal incentive contracts. *RAND Journal of Economics,* 33(4): 650-71

[38] Prendergast, C. (2008). Intrinsic motivation and incentives. *American Economic Review,* 98(2): 201-05

[39] Raymond, E. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* Cambridge: O'Reilly Press.

[40] 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

[41] Rousseau, D., and McLean Parks, J. (1993). The contracts of individuals and organizations. *Research In Organizational Behavior,* 15: 1-43

[42] Ryan, R. (1995). Psychological needs and the facilitation of integrative processes. *Journal of Personality,* 63: 397-427

[43] Ryan, R., and Deci, E. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist,* 55(1): 68-78

[44] Schein, E. (1965). *Organization Psychology.* Prentice-Hall: Englewood Cliffs, NJ.

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

[46] Skinner, B. (1953). *Science and Human Behavior.* New York: Macmillan.

TABLE 1. BREAKDOWN OF CONTRIBUTIONS: DEVELOPER LEVEL

	(1)	(2)	(3)	(4)	(5)	(6)
	# Developers	# Contributions	# Receiving Projects	Contributions per Developer (2)/(1)	Contributions per Project (2)/(3)	Project Membership per Developer
External Contributors						
Anonymous	NA	18,722	1,939	NA	9.7	NA
Non-members	15,133	32,293	4,071	2.1	7.9	NA
Highly restrictive	3,211	14,326	2,026	4.5	7.1	1.5
Mixed	2,308	25,802	1,784	11.2	14.5	2.9
Unrestrictive	1,860	12,569	1,356	6.8	9.3	2.0
Internal Contributors						
Highly restrictive	1,184	11,213	581	9.5	19.3	1.5
Mixed	1,026	21,769	592	21.2	36.8	3.0
Unrestrictive	474	9,822	277	20.7	35.5	2.7

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; Highly restrictive – developers who are only members of projects with highly restrictive licenses; Unrestrictive – 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 number of members.

TABLE 2. DISTRIBUTION OF CODE CONTRIBUTIONS BY DEVELOPER TYPE AND RECEIVING PROJECT CHARACTERISTICS (%)

	Contributing developers				
	(1)	(2)	(3)	(4)	(5)
	Anonymous	Highly Restrictive	Unrestrictive	Mixed	Non-members
<u>License Type</u>					
Highly restrictive	81.3	67.0	37.6	45.6	57.5
Restrictive	9.9	16.9	26.0	35.0	21.2
Unrestrictive	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. Project size is defined as number of members.

TABLE 3. BASELINE SPECIFICATION

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

	Developer type				
	(1)	(2)	(3)	(4)	(5)
	Anonymous	Open	Mixed	Closed	Non-members
Project license type:					
Highly Open (HO)	2.19 (2.34)	0.00	0.47 (0.82)	-2.93** (0.46)	5.08** (1.42)
Open (O)	-2.20** (0.70)	-1.62** (0.57)	0.99 (1.18)	-2.03** (0.60)	2.10* (1.16)
Closed (C)	-2.95** (0.38)	-1.96** (0.41)	0.59** (0.97)	-1.40** (0.62)	-2.65** (1.07)
ln(Number of members)	2.66** (0.45)	2.69** (0.047)	2.97** (0.46)	3.57** (0.49)	3.22** (0.42)
Average # contributions per cell	12.1	9.3	16.7	8.2	21.0
Hypotheses tests: Sorting on license type					
H_0: HR=R=UR	p<0.001	p<0.001	p<0.001	p<0.001	p<0.001
H_0: HR=UR	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 the 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: 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 (0.016). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

TABLE 4. 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
Project license type:					
Highly Open (HO)	3.56** (1.14)	0.00	0.54 (0.70)	-2.95** (0.35)	5.08** (1.23)
Open (O)	-1.03** (0.67)	-1.39** (0.45)	0.35 (0.83)	-2.30** (0.45)	2.78* (1.23)
Closed (C)	-2.06** (0.50)	-1.83** (0.36)	0.27 (0.76)	-1.72** (0.52)	-3.11** (1.19)
Project intended audience:					
Developer Tools	-2.81** (0.39)	-1.47** (0.057)	0.46** (0.85)	2.05** (0.72)	-1.65** (0.54)
End Users	-1.56** (0.62)	0.65** (0.99)	-1.28** (0.66)	0.14** (0.67)	0.53 (0.96)
ln(Number of members)	2.47** (0.36)	2.65** (0.030)	2.95** (0.31)	3.58** (0.33)	3.14** (0.32)
Average # contributions per cell	12.1	9.3	16.7	8.2	21.0
Hypotheses tests: Sorting on license type					
H_0: HR=R=UR	p <0.001	p<0.001	p<0.001	p<0.001	p<0.001
H_0: HR=UR	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, operating system, and programming language. We also include a linear control for the number of projects in the cell. We reject the following hypotheses: 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). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

TABLE 5. 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
Hypotheses tests: Sorting on license type					
H_0: HR=R=UR	p<0.001	p<0.001	p=0.52	p<0.001	p=0.11
H_0: HR=UR	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). Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

TABLE 6. PATTERN OF CONTRIBUTIONS BY CORPORATE SPONSORSHIP (in percent)

	(1)	(2)	(3)	(4)	(5)	(6)
	Number of projects	Anonymous	Open	Mixed	Closed	Non-members
Corporate Sponsorship	73	85.8	54.8	35.7	21.0	21.3
Not-for-profit	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 contributions for projects with strong corporate involvement (Corporate Sponsorship) and for not-for-profit projects.

TABLE 7. 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: base 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
			(0.148)	(0.171)
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 that 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 ntn-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 8. PATTERN OF RECIPROCITY

	(1)	(2)	(3)	(4)	(5)
Receiving license type:	# of projects	% of projects with reciprocal contributions	% of contributions received by reciprocal projects	% of contributions made by reciprocal projects	% of contributions from projects in (1) that are reciprocal
All	256	4.9	37.0	22.7	44.6
Highly Open (HO)	142	4.0	20.6	23.4	36.0
Open (O)	45	5.5	46.0	8.0	39.3
Closed (C)	69	7.5	74.8	52.0	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 9. DETERMINANTS OF RECIPROCITY

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

	(1)	(2)	(3)
Dummy for matching on license type:			
All licenses	0.120**	0.109**	
	(0.027)	(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**	0.045**	0.048**
	(0.019)	(0.016)	(0.047)
Dummy for matching on Programming Language	0.012	-0.037	-0.035
	(0.023)	(0.021)	(0.020)
Dummy for matching on Operating Systems	0.049	0.030	0.032
	(0.027)	(0.022)	(0.022)
ln(Number of members), receiving		0.003**	0.003
		(0.001)	(0.001)
ln(Number of members), contributing		-0.001	-0.001
		(0.002)	(0.002)
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 10. CONTRIBUTIONS AND PROJECT PERFORMANCE

Dependent variable: ln(Number of Project Downloads).
Non-linear Least Squares (N=37,833)

	(1)	(2)	(3)
Elasticity: *Total Contributions Stock*	0.307**	0.305**	0.309**
	(0.007)	(0.007)	(0.008)
Marginal Productivity: *Non-member Contributions* (normalization)		1.000	1.000
Relative Marginal Productivity: *Internal Contributions*		0.344**	0.222**
		(0.143)	(0.065)
Relative Marginal Productivity: *External Contributions*		0.620**	
		(0.138)	
Relative Marginal Productivity: *Matched Contributions*			0.174**
			(0.033)
Relative Marginal Productivity: *Unmatched Contributions*			0.169**
			(0.028)
ln(Number of members)	0.015**	0.015**	0.015**
	(0.001)	(0.001)	(0.001)
Test: Marginal Productivity Internal=Marginal Producitivity External (p-value)	NA	0.08	NA
Test: Marginal Productivity Matched=Unmatched Marginal Productivity (p-value)	NA	NA	0.88
Adjusted-R^2	0.544	0.544	0.542

Notes: This "non-members" 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. Standard errors are robust to arbitrary heteroskedasticity. ** significant at 1%, * significant at 5%.

CENTRE FOR ECONOMIC PERFORMANCE
Recent Discussion Papers

892	Guy Michaels Ferdinand Rauch Stephen J. Redding	Urbanization and Structural Transformation
891	Nicholas Bloom Christos Genakos Ralf Martin Raffaella Sadun	Modern Management: Good for the Environment or Just Hot Air?
890	Paul Dolan Robert Metcalfe	Comparing willingness-to-pay and subjective well- being in the context of non-market goods
889	Alberto Galasso Mark Schankerman	Patent Thickets and the Market for Innovation: Evidence from Settlement of Patent Disputes
888	Raffaella Sadun	Does Planning Regulation Protect Independent Retailers?
887	Bernardo Guimaraes Kevin Sheedy	Sales and Monetary Policy
886	Andrew E. Clark David Masclet Marie-Claire Villeval	Effort and Comparison Income Experimental and Survey Evidence
885	Alex Bryson Richard B. Freeman	How Does Shared Capitalism Affect Economic Performance in the UK?
884	Paul Willman Rafael Gomez Alex Bryson	Trading Places: Employers, Unions and the Manufacture of Voice
883	Jang Ping Thia	The Impact of Trade on Aggregate Productivity and Welfare with Heterogeneous Firms and Business Cycle Uncertainty
882	Richard B. Freeman	When Workers Share in Profits: Effort and Responses to Shirking
881	Alex Bryson Michael White	Organizational Commitment: Do Workplace Practices Matter?
880	Mariano Bosch Marco Manacorda	Minimum Wages and Earnings Inequality in Urban Mexico. Revisiting the Evidence
879	Alejandro Cuñat Christian Fons-Rosen	Relative Factor Endowments and International Portfolio Choice
878	Marco Manacorda	The Cost of Grade Retention

877	Ralph Ossa	A 'New Trade' Theory of GATT/WTO Negotiations
876	Monique Ebell Albrecht Ritschl	Real Origins of the Great Depression: Monopoly Power, Unions and the American Business Cycle in the 1920s
875	Jang Ping Thia	Evolution of Locations, Specialisation and Factor Returns with Two Distinct Waves of Globalisation
874	Monique Ebell Christian Haefke	Product Market Deregulation and the U.S. Employment Miracle
873	Monique Ebell	Resurrecting the Participation Margin
872	Giovanni Olivei Silvana Tenreyro	Wage Setting Patterns and Monetary Policy: International Evidence
871	Bernardo Guimaraes	Vulnerability of Currency Pegs: Evidence from Brazil
870	Nikolaus Wolf	Was Germany Ever United? Evidence from Intra- and International Trade 1885 - 1993
869	L. Rachel Ngai Roberto M. Samaniego	Mapping Prices into Productivity in Multisector Growth Models
868	Antoni Estevadeordal Caroline Freund Emanuel Ornelas	Does Regionalism Affect Trade Liberalization towards Non-Members?
867	Alex Bryson Harald Dale-Olsen	A Tale of Two Countries: Unions, Closures and Growth in Britain and Norway
866	Arunish Chawla	Multinational Firms, Monopolistic Competition and Foreign Investment Uncertainty
865	Niko Matouschek Paolo Ramezzana Frédéric Robert-Nicoud	Labor Market Reforms, Job Instability, and the Flexibility of the Employment Relationship
864	David G. Blanchflower Alex Bryson	Union Decline in Britain
863	Francesco Giavazzi Michael McMahon	Policy Uncertainty and Precautionary Savings

The Centre for Economic Performance Publications Unit
Tel 020 7955 7284 Fax 020 7955 7595 Email info@cep.lse.ac.uk
Web site http://cep.lse.ac.uk