EDS Innovation Research Programme
DISCUSSION PAPER SERIES

No.019

Intellectual Property, Technology and Productivity

Motivation and Sorting in Open Source Software Innovation

Sharon Belenzon
Mark Schankerman

November 2008

EDS an HP company

LSE THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

EDS Innovation Research Programme

Is a collaboration between EDS and leading LSE academics from a range of disciplines researching the determinants of innovation, technology, creativity and productivity and the policies needed to foster them.

The Discussion Paper series features the research of the four teams;

1. Public policy and services (Patrick Dunleavy, Department of Government)

2. Intellectual property, technology and productivity (John Van Reenen, Danny Quah, Centre for Economic Performance & Department of Economics)

3. Media, connectivity, literacies and ethics (Robin Mansell, Department of Media & Communications)

4. Complexity, mediation and facilitation. (Patrick Humphreys, Institute of Social Psychology)

Published by

EDS Innovation Research Programme

London School of Economics and Political Science

Houghton Street

London WC2A 2AE

Motivation and Sorting in Open Source Software Innovation

Sharon Belenzon
Fuqua School of Business, Duke University
Mark Schankerman
University of Arizona, Centre for Economic Performance and London School of Economics

Abstract

This paper studies the role of intrinsic motivation, reputation and reciprocity in driving open source software innovation. We exploit the observed pattern of contributions – the 'revealed preference' of developers – to infer the underlying incentives. Using detailed information on code contributions and project membership, we classify developers into distinct groups and study how contributions from each developer type vary by license (contract) type and other project characteristics. The central empirical finding is that developers strongly sort by license type, project size and corporate sponsorship. This evidence confirms the importance of heterogeneous motivations, specifically a key role for motivated agents and reputation, but less for reciprocity.

1. Introduction

This paper studies how incentives and sorting behavior affect open source software innovation. In open source, the source code is made available for public use and development under specific conditions that depend on the license governing the project. Programmers who contribute to open source projects are typically unpaid, though corporate sponsorship and financing have increased sharply in recent years. This raises an apparent paradox: How is open source innovation sustained in the face of free-rider problems and the absence of direct monetary compensation? This issue is important not only in the software sector, but also other areas in which 'open commons' production has been proposed, including data bases, biotechnology and nanotechnology.[1]

There are four main competing explanations in the literature. First, contributions may be viewed as an investment in the programmer's reputation which yields payoffs in the form of peer recognition (Raymond, 2001) or commercial rewards in the labor market (Lerner and Tirole, 2001, 2002; Johnson, 2002, 2004). Second, developers may expect to gain later from reciprocal contributions from projects to whom they have previously contributed (Raymond, 2001; Lakhani and von Hippel, 2003). Third, developers may be intrinsically motivated to contribute by their strong identification with the 'ideology' underlying the open source movement (Raymond, 2001). This ideology, whose origin is in the earlier 'free software' movement associated with Stallman at MIT, is strongly tied to maintaining 'highly restrictive' licenses that effectively prevent any commercial use of the source code.[2] Lastly, developers may get pure utility value or learning benefits from participation (put another way, their marginal utility of effort may be positive over some range – Kreps, 1997; Glazer, 2004). The first two explanations involve extrinsic incentives – even though they are not embodied in explicit employment contracts between the contributor and open source project – whereas the other two arguments require some form of intrinsic motivation.

There are a number of previous empirical studies of motivations in open source development, but they suffer from two important limitations. First, these studies are typically based on small sample surveys of programmers (e.g. Haruvy, Wu and Chakravarty, 2003; Hertel,

[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, Bryan 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.

Krishnan and Slaughter, 2003). More importantly, they rely exclusively on what programmers say their motivations are, without any way to corroborate these 'announced preferences'. One notable exception is Hann et. al. (2004), who test the labor market hypothesis of Lerner and Tirole (2002) by studying the relationship between wages of developers and their contributions to Apache, a major open source project. They find that wages are related to the contributing developers' rankings by the Apache Software Foundation, but not to the volume of their contributons, suggesting that contributions may be motivated more by labor market signalling than by human capital accumulation.

The aim of this paper is to determine the extent to which intrinsic motivation, reputation and reciprocity motives drive open source innovation. The empirical analysis is based on a large scale data set with detailed information about software code contributions to open source projects, including the license type, size, programming language, operating system and intended audience for both the contributing and receiving projects. We study the empirical determinants of software code contributions by focusing on distinct groups of developers. We classify each developer according to the type of open source license governing the projects of which she is a registered member. This yields four developer categories: highly restrictive, unrestrictive, mixed, and anonymous. We investigate how the pattern of contributions from each developer type varies across the license type and other characteristics of the contributing and receiving projects. The key innovation in our approach is that we exploit the observed pattern of contributions – the *'revealed preference' of developers* – to infer the underlying incentives.

Our econometric approach builds on the methodology used in the literature on patent citations. Until recently, that literature has focused exclusively on the determinants of citations at the patent level, rather than the inventor level, because of the difficulty in identifying inventor names (Jaffe and Trajtenberg, 2005).[3] We are able to avoid this problem because our data set includes a unique identification number for each contributing developer. Our methodology is to aggregate code contributions into cells defined by a set of characteristics of the contributing and receiving projects, and then to use these cells as the observations in the estimation procedure. The key identification assumption is that the license type and other characteristics of contributing and receiving projects are exogenous with respect to the decisions of individual developers

[3]Manuel Trajtenberg has pioneered efforts to construct inventor-level data, and a number of recent papers explore patent citation flows at this micro-level. See Trajtenberg et. al. (2006).

to contribute. While we cannot rule out unobserved developer or project heterogeneity, our focus is on the *interactions* between the contributing developer type and project characteristics (not on the level effects of these characteristics), and such heterogeneity would induce bias only if it is correlated with these interactions.

The central empirical finding in this paper is that developers strongly sort on a variety of observed project characteristics. We interpret this finding as showing that software developers are heterogeneous with respect to their motivations. *First,* we find assortative matching on the project license type. Highly restrictive developers almost exclusively contribute to projects with highly restrictive licenses, indicating an important role for 'motivated agents' – developers dedicated to the ideology of the open source movement. Unrestrictive developers primarily contribute to projects with unrestrictive (more commerical) licenses. If unrestrictive 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,* developers from unrestrictive projects are more likely to contribute to larger projects and to those that are sponsored by corporations. This evidence confirms that labor market reputation (career concerns) plays an important role, as emphasized by Lerner and Tirole (2002). At the same time, however, we find that the size of the receiving project also matters for other developer types, albeit to a lesser extent, which indicates that the peer recognition motive also plays a role. *Third,* restrictive developers are much more likely to contribute to projects aimed at end users (e.g. computer games), whereas unrestrictive developers target developer oriented (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 motivated agents is more a substitute for proprietary software innovation on the end-product side.[4] All three of these findings on sorting behavior are robust to various empirical specifications including a wide range of control variables.

Finally, 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 unrestrictive (commercially oriented) developers than for highly restrictive developers (motivated

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

agents). This suggests that reciprocity is associated more with building reputations than with intrinsic motivation. Lastly, while reciprocity accounts for a large fraction of contributions for a few projects, the vast majority of projects does not involve such behaviour.

We believe our empirical findings — in particular, the heterogeneity of motivations and the strong sorting behavior it induces – are relevant for thinking about employment contract design in other contexts. 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. 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.[5] Among other things, this literature shows that optimal incentives depends strongly on the form, and heterogeneity, of worker motivation. When instrinsic motivation is strong (e.g., the academic and NGO sectors), low-powered incentives may be more efficient for the principal. Frey (1997) shows that extrinsic incentives may even crowd out intrinsic motivation (for an empirical test, see Frey and Oberholzer-Gee, 1997). In addition, the sorting behavior that can arise from heterogeneous motivations, and other characteristics of the agent, is important to take into account in econometric studies of contract design (Ackerberg and Botticini, 2002).

Despite the progress on the theory, however, there are very few empirical studies of motivation and contracts. In large part, this is due to the difficulty of identifying different worker 'types' in a way that allows one to study how they sort across contracts and the implications for performance outcomes. In this paper, we are able to sidestep the problem by characterising software developers *a priori* on the basis of the open source projects of which they are members. The key to doing this is our use of data at the *individual developer level.* This allows us to study the empirical pattern of contributions by each developer type and to infer from this pattern their underlying motivations.

While there are no formal employment contracts between contributing developers and open source projects, the project license is itself a contract governing the subsequent use and openness

[5]Leading examples include Frey (1997), Kreps (1997), Francois (2000), Murdock (2002), Benabou and Tirole (2003), Delfgauuw and Perez-Castrillo (2004), Besley and Ghatak (2005), Delfgauuw and Dur (2007, 2008), and Prendergast (2008).

of the contributed software code. Thus our paper can be viewed as a contribution to the empirical literature on motivation and contracts, in that we show that contract design has strong sorting effects. In this paper we treat the contract choice as exogenous. Lerner and Tirole (2002) study the choice of open source license, arguing that the relevant tradeoff is between greater proprietary control with the more commerical, unrestrictive licenses and a potentially greater pool of contributors with more restrictive licenses. 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 contract choice and the resulting sorting effects into one empirical framework.

The paper is organised as follows: Section 2 describes the data set and the key variables used in the empirical analysis. Section 3 sets out the main hypotheses about developer motivations for open source innovation. Section 4 presents descriptive evidence on the main features of the patterns of developer contributions. Section 5 presents the econometric framework. The empirical results and their implications are discussed in Section 6. Brief concluding remarks follow.

2. 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 a specialised software algorithm that accessed each project registered on SourceForge and extracted all available information about the project, the participating developers and their software code contributions. We launched this software in November 2005 and collected detailed information on all 61,514 open source projects listed on SourceForge and all code contributions ('patches') made to these projects.[6] The data set covers the period 1999-2005.

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

[6] Table A.2 in the appendix provide information about the most active receiving and contributing projects in our dataset.

the project license, which focuses mainly on the extent to which commercial use is allowed. More restrictive licenses constrain such use more severely. 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).[7] 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 restrictive (HR): 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) Restrictive (R)***:** 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 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) Unrestrictive (UR): 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 (this applies to 7 percent of projects), we classify the project as highly restrictive if at least one of its licenses is highly restrictive, and as unrestrictive if all of its licenses are unrestrictive.[8] The remaining projects are classified as restrictive. In the complete sample of 61,514 projects, 65 percent operate under a highly restrictive license and 13 percent under an unrestrictive license.[9] A

[7] For a good discussion of different license types and their restrictions, see Lerner and Tirole (2002). Table A.1 in the appendix to this paper summarizes the conditions defining each type and provides some examples.

[8] The results reported in the paper are robust to alternative assumptions, such as classifying projects as highly restrictive only if all of their licenses are highly restrictive, or classifying projects as unrestrictive only if the majority of their licenses are unrestrictive.

[9] 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.

project can receive contributions from its members or developers who are not affiliated with the focal project. In what follows, we focus the analysis on the more interesting case where contributions are made by developers who have no formal affiliations with the receiving project.[10]

Among projects that receive at least one code contribution, 60 percent operate under HR licenses, and 20 percent under UR licenses. This difference reflects that fact that highly restrictive projects receive fewer contributions per project than those under unrestrictive licenses.

Developer Type: The developer type is based on the projects of which she is a member. We classify a developer as highly restrictive if all of the projects to which she belongs operate under highly restrictive licenses, as defined above. We define a developer as unrestrictive if all of the projects to which she belongs operate under unrestrictive licenses. All other developers who are members of projects are classified as Mixed.[11] We include two other categories: 'Anonymous' – developers who submit patches without revealing their identity to the receiving project manager or members, though the contribution itself is made public; and 'Non-members' – contributors who are not registered as members of any project.

Size of Project: The size of the project is defined by the number of developers registered as formal members, including the project manager.[12]

Resoultion of the code contribution: From information on SourceForge, we know whether the open source projects reveal to outsiders that individual code contributons have been accepted (i.e., incorporated ino the project software). About 75 percent of projects registered on SourceForge reveal the outcome of (at least some) contributions.

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.[13]

[10] In the overall sample, 39 percent of contributions are made by project members, i.e., these contributions are internal to the receiving project. The remaining 61 percent are from developers that are either members of other projects, not members of any project, or do not reveal their identity when making their contribution (anonymous). The percentage of internal contributions is higher for unrestrictive projects than for highly restrictive projects (50 and 34 percent, respectively).

[11] On average, developers belong to 1.28 projects that either receive or contribute patches. Members of unrestrictive projects are somewhat more diversified than those of highly restrictive projects – they belong to 1.5 and 1.3 projects, on average, respectively.

[12] The alternative way to measure size would be to use the total number of patches received by the project. Since we want to explain the pattern of code contributions by developers, it would be problematic to treat this measure as exogenous for small projects since the developer's decision to contribute may have a non-negligible effect on project size.

[13] The programming languages included in each of the five categories are as follows:

Operating System: Each project is conducted on one or more operating system, which is the platform on which the program runs. We use four systems, roughly following Lerner and Tirole (2002): Microsoft, Open Source Independent, POSIX, and Mixed[14].

Intended Audience: Each software project is listed as targeted to different intended audiences (user groups). We group the 19 audience types in SourceForge into five categories: developers (programming tools), end users (desk top applications, computer games etc), system administrators, mixed (of the preceding three), and other[15]. About 30 percent of receiving projects are developer oriented and 18 percent are end user oriented.

3. Motivations behind Open Source Innovation

We focus on four main motivations: (1) Motivated agents, (2) Reputation, (3) Reciprocity, and (4) Utility/Learning. The literature and available small sample survey evidence suggest a role for each of these motivations in open source developement (e.g. Haruvy, Wu and Chakravarty, 2003; Lakhani and von Hippel, 2003; Hertel, Krishnan and Slaughter, 2003; Lerner and Tirole, 2002).

Motivated agents: Following Francois (2000) and Besley and Ghatak (2005), 'motivated agents' are those whose utility from participation is greater when they match to employers of their own type. In our context, 'type' refers to the restrictiveness of the license associated with the open source project. It has been claimed that many software developers have a strong (ideological) preference for maintaining fully open source code in all software embodying it, i.e. highly restrictive GPL licenses. This was the original driving force behind the 'free software'

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.

[14]In addition to Lerner and Tirole (2002), we use Wikipedia and http://osapa.org/wiki/index.php/SF/Freshmean Trove to group operating systems to the four main categories. The POSIX category includes Linux, Solaris and BSD; OS independent includes only OS independent operating systems; Microsoft includes all of Microsoft's operating systems (such as MS-DOS and WinXP). The remaining category includes software that operates on more than one operating system.

[15]The End Users category includes end users/desktop and advanced end users. The 'Other' category, which account for about 4 percent of the projects, includes mainly aerospace, education, science/research and healthcare.

movement (Raymond, 2001). In economic terms, this means that the utility a highly restrictive developer gets from making a code contribution is larger if it is made to a highly restrictive project (in the extreme form, it is zero if made to a project with any other license type). Thus, the motivated agents hypothesis predicts positive (but not necessarily exclusive) sorting of highly restrictive developers to projects with highly restrictive licenses. This hypothesis makes no prediction about sorting of unrestrictive developers.

Reputation: There are two distinct types of reputation that may be important: commercial reputation and peer recognition. Lerner and Tirole (2002) argue that developers improve their labor market prospects by signalling their quality through participation in open source projects. These signalling benefits are likely to be greater: (1) when the project to which they contribute is larger and hence more visible (Johnson, 2002), (2) when the project reveals the outcome of the contribution (i.e., whether it has been accepted by the project manger), (3) when the project is sponsored by commercial firms, and (4) when the project is aimed at developers (programming tools) rather than end users. The prediction of the commercial reputation (labor market signalling) hypothesis is that unrestrictive developers should sort positively on these four dimensions.

The second type of reputation gain is peer recognition, 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. The peer recognition hypothesis predicts sorting on these three dimensions, but this motivation should be unrelated to commercial sponsorship of projects. Finally, neither the commercial reputation nor peer recognition motivation should operate for anonymous contributors.

One last point concerns sorting by unrestrictive developers on license type. If unrestrictive developers understand that assortative matching in the population occurs, then the (commercial) reputation hypothesis predicts that these developers would tend to sort on projects with unrestrictive licenses. That is, they would tend to contribute to projects where reputation gains are most likely to be obtained.

Reciprocity: Developers who are members of a project may make contributions in response to (or anticipation of) contributions made by other developers to their project.[16] This hypothesis

[16]In principle, reciprocity can be sustained as an equilibrium in a repeated game setting, though in practice it may be difficult in the open source context to detect deviation (i.e., knowing what level of contribution reveals non-deviation) and to punish it.

implies that contributions from project i to project j in year t should be related to whether project j has contributed to project i prior to year t. Notice that the reciprocity motive should not apply to anonymous contributors or to contributors who are not members of any projects.

Utility/learning: Developers may contribute code to projects because they enjoy doing so and/or because they learn from the process.[17] The learning benefits from making contributions are likely to be stronger if the contributing and receiving projects use the same programming language or operating system. Thus, the utility/learning hypothesis predicts sorting on these dimensions, but no sorting on license type.

Costs of contributions: Developers incur effort costs in making code contributions, and are more likely to contribute when these costs are lower. This should be the case when the programming languages, and possibly the intended audience, of the contributing and receiving project are similar. Note that these predictions are essentially the same as for the utility/learning hypothesis, so it is not possible to distinguish empirically between the role of effort costs and utility/learning benefits.

4. Descriptive Statistics

Table 1 summarizes characteristics of *'receiving projects'* (those that receive at least one code contribution). Notice first that the distribution of code contributions is highly concentrated. Excluding contributions by developers affiliated with the receiving project, only five percent of projects receive *any* contributions over the ten years covered by the data.[18]. Second, most receiving projects are small – 90 percent have 10 or fewer members. The mean number of received contributions rises monotonically, and very strongly, with project size. In part this may reflect correlation between unobserved project quality (which affects contributions) and project size, but it is likely also to reflect the reputation incentive of contributors, as discussed earlier.

Third, project license type is correlated with other project characteristics. highly restrictive projects receive fewer contributions than unrestrictive projects (13.1 and 20.7 patches, respectively). Highly restrictive projects are also much more likely to focus on end user software as

[17] In addition, they may hope to influence the direction of the software project in ways from which they expect to benefit.

[18] Even if we include internal contributions, the figure is only 6.1 percent.

opposed to developer oriented programming tools (not shown in table). Of the highly restrictive projects, 25 percent focus on end users and 16 percent on developer tools; for unrestrictive projects the figures are 10 and 42 percent, respectively. License type is not systematically related to programming language or operating system (about 7 percent of both highly restrictive and unrestrictive projects adopt Microsoft as their operating system, with the remaining projects mostly adopting OS Independent and POSIX).

Table 2 summarizes characteristics of *'contributing projects'* that made at least one code contribution to another project.[19] As with receiving projects, we observe that contribution activity is very concentrated. Only about 10 percent of projects make any contributions to other projects over the ten years of the sample. There is variation across license types in the pattern of contributions. Highly restrictive projects make fewer contributons than other license types, despite the fact that they are larger, and they are more focused on end user software, as compared to unrestrictive projects. The mean number of contributions also rises with project size, but much less sharply than for receiving projects.

In Table 3 we examine how the level of contribution activity varies with individual developer (rather than project) type. Of the 113,191 developers listed on SourceForge, only 14 percent make any contributions, and only 11.5 percent make contributions to projects with which they are not affiliated. Moreover, the distribution of contributions by developers is highly skewed. In total, highly restrictive developers account for four times as many contributions as unrestrictive developers (16 versus 4 percent), though the mean number of contributions is very similar for the two groups. Anonymous contributors account for about 10 percent. If we made the strong assumption that anonymous contributors and highly restrictive developers are pure motivated agents, we would conclude that this motivation accounts for as much as 25 percent of total contributions. But this is an upper bound, since we will show that the contribution patterns of these developer types indicate that they have mixed motivations.

Finally, we examine how the pattern of contributions varies with the developer type and

[19] We treat a project as contributing if at least one of its members made one or more contributions to a different project. Since a developer can be a member of more than one project, we include only contributions where the contributing developer is not a member of the receiving project as well.

receiving project characteristics. Table 4 identifies three key aspects of sorting behavior that will be confirmed in the subsequent econometric analysis. First, anonymous and highly restrictive developers are much more likely to contribute to projects with highly restrictive licenses than other types, while unrestrictive developers focus more heavily on projects with unrestrictive licenses. Second, anonymous and highly restrictive developers focus much more heavily on projects targeted at end users, while unrestrictive developers are more likely to contribute to projects aimed at developer programming tools. This 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, unrestrictive developers are much more likely to contribute to the very large projects (>50 members) than anonymous or highly restrictive developers. These last two findings are consistent with the hypothesis that unrestrictive developers are more driven by commerical motives, since programming skills are likely to be more effectively signalled by contributing to developer tools and larger projects.

5. Econometric Specification

We want to estimate the effect of project characteristics on the pattern of code contributions.[20] To do this, we adopt the approach used in the literature on patent citations. We 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. This allows us to estimate the effect of cell characteristics (and interactions) on the number and pattern of patches. In these regressions, we control for the number of potentially contributing and receiving projects in the same cell, where the latter include all projects in the data (including those that receive zero contributions).

Since the number of patches is an integer, we use an econometric specification for count data. The general model is

$$E(y_{c,r}|\mathbf{x}_{c,r}) = G(\mathbf{x}_c, \mathbf{x}_r) \qquad ((1))$$

[20] A contributing project is defined as the project of which the contributing developer is a member. If the developer belongs to more than one, we treat each separate project as a contributing project. As before, we exclude contributions by developers to projects of which they are a member.

where, $y_{c,r}$ denotes the number of contributions made by developers who are members of projects in cell c to projects in cell r, $\mathbf{x}_c$ is a vector of characteristics of the contributing project c (including the developer type), $\mathbf{x}_r$ is a vector of characteristics of the receiving project r, and G is a functional form which we need to assume. The main choice for G is between Poisson and Negative Binomial models. The Poisson model imposes the strong restriction $Var(y_{c,r}|\cdot) = E(y_{c,r}|\cdot)$; the Negative Binomial allows for 'over-dispersion' which is a common feature of count data. But this generality comes at a cost – if the specification of the conditional variance in the Negative Binomial is wrong, the parameter estimates of the condition mean are inconsistent, whereas the Poisson estimates of the conditional mean are consistent even with over-dispersion. For this reason we adopt the Poisson specification. To allow for over-dispersion, we use the Poisson QMLE estimation procedure with variance specified as $Var(y_{c,r}|\cdot) = \lambda^2 E(y_{c,r}|\cdot)$.

An observation is defined as a combination of characteristics of the contributing and receiving projects. We study five contributing project characteristics: developer type (t), intended audience (u), number of members (project size, m), programming language (p), and operating system (o). We denote the vector of contributing project characteristics by $c = (t, u, m, p, o)$. We study five characteristics of receiving projects: license type (L), intended audience (U), number of members (project size, M), programming language (P), operating system (O), and year of registration (Y). We let $r = (L, U, M, P, O, Y)$.

The baseline specification is

$$y_{c,r} = G(\boldsymbol{\alpha}\mathbf{x}_c + \beta\mathbf{x}_r + \boldsymbol{\gamma}\mathbf{x_c}\mathbf{x_r}) + \varepsilon_{c,r} \qquad ((2))$$

where $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ denotes the vector of parameters to be estimated and we assume $E(\varepsilon_{c,r}|\mathbf{x}_c, \mathbf{x}_r) = 0$. In this specification, we focus on the interactions between the developer type and the characteristics of the receiving project, controlling for a full set of linear dummies to capture both the contributing and receiving project characteristics.[21] Our main objective is to determine whether there is sorting of developers according to the license type of receiving projects (motivated agents) and project size (reputation), which is revealed by the coefficients on the set of interaction variables. We also test whether there is sorting of developers on other dimensions of the contributing and receving projects. In the set of duumy variables for interactions between

[21] As explained earlier, we do not directly observe the developer type. We infer it from the project affiliations (membership) of the developer.

developer and project license types, we drop the dummy for HR developer-HR license type. All other interaction coefficients measure impacts relative to this reference group.

The key identifcation assumption is that the license type and other characteristics of contributing and receiving projects are exogenous with respect to the decisions of individual developers to contribute. The main source of concern is that unobserved developer characteristics, especially quality, may induce correlation. For example, high quality developers may sort on some dimensions of contributing or receiving projects. But our primary focus is on the *interactions* between the contributing developer type and project characteristics, the $\boldsymbol{\gamma}$ coefficients in equation (2) – not on level effects given by $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. Unobserved heterogeneity will induce bias only if it is correlated with these interactions. It is hard to assess *a priori* what direction any such bias might take (i.e., how developer quality might be correlated with specific project characteristics).

6. Econometric Results

6.1. Baseline Specification

Table 5 presents the parameter estimates for the interaction terms between the type of the contributing developer and the license and size of the receiving project. The specification also includes a complete set of linear dummy variables for receiving project characteristics – year of registration, intended audience, operating system, and programming language. The empirical results confirm strong sorting behaviour by different types of developers. First, highly restrictive developers are much more likely to contribute to projects that have highly restrictive licenses and least likely to contribute to projects with unrestrictive licenses. The $HR-HR$ coefficient is normalised to zero, while the interaction coefficients of highly restrictive developers with mixed and unrestrictive projects are -0.858 and -1.38, respectively. This means that the marginal effect of changing a project's licsense type from highly restrictive to unrestrictive is to reduce the number of contributions by highly restricive developers by 1.38, which is nearly a third of the mean number of contributions by such developers. We strongly reject the hypothesis that there is no sorting by license type for these highly restrictive developers ($p - value < .001$).

Contributions by anonymous developers cannot be driven by career concerns because they do not reveal their identity and cannot gain peer recognition. Thus anonymous contribution activity indicates either the importance of motivated agents or pure utility value/learning from

contributing. If they are primarily motivated agents, we expect them to sort on highly restrictive projects, whereas the utility value/learning incentive predicts no systematic sorting on license type. Table 5 shows that anonymous developers sort in a way similar to highly restrictive developers – they are more likely to contribute to highly restrictive projects than to either mixed or unrestrictive ones. Moving from a highly restrictive license to an unrestrictive one reduces the number of contributions by anonymous developers by 16 percent (0.55/3.21). These results for highly restrictive and anonymous developers support the motivated agents hypothesis.

For unrestrictive developers we also find strong sorting but of the opposite pattern. They are much more likely to contribute to unestrictive projects, as compared to highly restrictive ones (compare the coefficients on the interaction with unrestrictive projects, -2.22, with the one on highly restricted projects, -3.08). These estimates imply that moving from an unrestrictive to a highly restrictive license reduces the contributions of unrestrictive developers by about 90 percent, evaluated at the means. We decisively reject the null hypothesis of no sorting by unrestrictive developers ($p - value < .001$).

We also find that the size of the receiving project affects the flow of incoming contributions. This holds for all developer types, but the largest impact is found for unrestrictive developers. A ten percent increase in the number of members (our measure of project size) raises the number of contributions by unrestrictive developers by 0.25, which is about 25 percent of the mean. The corresponding percentage increases are much smaller for the other developer types: 6.1 percent for anonymous, 4.2 percent for highly restrictive, and 2.1 percent for mixed. This finding indicates that reputation building plays a role for all developer types, but it is particularly important for the more commercially-oriented (unrestrictive) developers.[22]

6.2. Robustness Analysis

The first robustness check is to generalise the empirical specification by allowing for interaction effects (sorting) between the developer type and both intended audience and programming

[22]Two points should be noted. First, the fact that the estimated coefficient on project size is also significant for anonymous developers (for whom reputation gains do not apply) suggests that there is also a utility/learning benefit that may be related to the size of the project. Second, we also estimated the baseline specification in Table 5 separately for different receiving project sizes. The results show the same pattern of sorting between developer type and receiving project license, for each size category. The only difference is that for small receiving projects, anonymous developers exhibit no sorting on license type.

language of the receiving project. Table 6 presents the results (for brevity, we omit coefficients on the interactions between developer type and programming language, which have no intrinsic interest). All of the key findings about sorting on license type are preserved. The project size effects are also robust, confirming the special importance of the reputation incentive for unrestrictive developers. The new finding is that anonymous and highly restrictive developers are more likely to contribute to end user oriented projects, as compared to developer oriented projects. This is shown by the coefficients in rows (d) and (e). The opposite is true for unrestrictive developers. We easily reject the null hypothesis that there is no sorting by intended audience, for each developer type ($p - values < 0.001$).These findings confirm that the choice of license type and project orientation (a decision made by the project manager or sponsor) strongly affects the composition of contributing developers.[23]

Second, thus far we have included controls only for the characteristics of the *receiving* project. Table 7 expands the specification by adding a set of linear dummy variables for the intended audience and programming language of the *contributing* project. Since the analysis is based on cells defined by the interactive characteristics, we need to use more narrowly defined cells here (this accounts for the larger number of potential flow links - 'observations' - listed in the table). Unfortuantely, we cannot include anonymous contributors in this analysis because we cannot identify the relevant contributing project characteristics for those developers.

Including the characteristics of the contributing projects does not change our earlier findings about how developers sort on project license type. Contributions by highly restrictive developers are more more likely to go to projects with highly restrictive licenses, and the reverse holds for unrestrictive (and to a lesser extent, mixed) developers. The estimated marginal effects in Table 7 are much smaller than those in Table 6, but this is simply an artifact of having much more disaggregated cells. This is apparent from a comparison of the mean number of contributons by developer type in the two tables. In fact, the implied marginal effects from Table 7, expressed as a percentage of the mean contributions, are very similar to those in Table 6. For example, using Table 6, we find that moving from a highly restrictive to an unrestrictive license reduces contributions of highly restrictive developers by 28.6 percent ($-1.16/4.06 = 0.286$), evaluated at the mean. The corresponding figure from Table 7 is 29.1 percent ($-0.145/0.499 = 0.291$). In

[23]This was the assumption underlying the important paper by Lerner and Tirole (2002).

addition, we again find that all developer types sort on the size of the receiving projects, and that the contributions from unrestrictive developers are most sensitive to project size, consistent with their greater reliance on (commercial) reputation motivation.

Third, we generalise the specification given in Table 7 by adding dummy variables for whether there is matching on the intended audience and the programming language of the contributing and receiving projects. For brevity we do not present the full set of results. All of the key findings about sorting on license type and the effects of receiving project size are very similar to those presented in Table 7. The new finding is that the number of contributions is significantly greater when there is such matching. We began by introducing two dummy variables, one for whether there is any matching on intended audience (end users versus developer tools) and another for matching on programming language (there are four language groups). The estimated marginal effects (standard error) are 0.284 (0.011) and 0.087 (.007), respectively, which are equivalent to 7.1 and 2.2 percent of the mean number of contributions. We also estimated a specification in which we allow the effects of matching to be different for each type of intended audience and programming language. While we find that the impact of matching does vary across types, the estimated marginal effects are statistically very significant in every case.[24]

There are two explanations for the matching on intended audience and programming language. The first is that reputation building is likely to be more effective and more valuable among developers with similar 'tastes' and experiences. At the same, however, the utility and learning benefits for the contributing developer are also likely to be greater, and the effort cost of making contributions lower, when there is matching on these two dimensions. With the available data, we cannot distinguish between these two effects.

6.3. Extensions

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

Public Resolution of Contributions: The first extension exploits information on whether the

[24]The marginal effects (standard error) are 0.268 (.021) for developer tools and 0.155 (.024) for end users. For programming languages the estimates are 0.272 (.039) for Object Oriented, 0.443 (.028) for Imperative, 0.119 (.015) for Scriptive, and 0.947 (.084) for Dynamic. The reference category for intended audience and programming language is Other. For details of languages in each category, see note 13.

receiving project publishes whether the contribution made by a developer is accepted or rejected. After a developer makes a code contribution, the project manager (or members) decides whether to accept the new code and to make that decision public on SourceForge. While the decision is made for each contribution separately, projects differ systematically on the degree to which they publish these outcomes. In our sample, 1,991 projects that receive contributions have at least one 'closed' (resolved) patch.[25] For these projects, the average share of patches with reported resolution is 0.65 (median is 0.88).

We use this information to identify the reputation incentive more sharply. We expect developers motivated by commercial reputation to be much more likely to contribute to projects with public resolution than other developers. To investigate this hypothesis, we define a dummy variable equal to one for projects that reveal the outcome for at least 50 percent of the (closed) contributions they receive over the sample period.[26] With this threshold, 65.8 percent of the projects are identified as having public resolution. It is worth noting that they tend to be larger than those that do not publicly disclose outcomes (mean numbers of contributions received are 32.7.and 11.8, respectively).

Table 8 presents the results for the baseline specification where we include the dummy variable for public resolution. The key findings about sorting on license type and the effects of receiving project size are very similar to the baseline results in Table 5. The new finding is that public resolution strongly affects the flow of contributions. The estimated marginal effect is statistically significant for all developer groups, but the point estimate is about three times larger for unrestrictive developers. Adding public resolution raises contributions by unrestrictive developers by 2.77, which is twice as large as their average number of contributions. To put it another way, unrestrictive developers very rarely contribute to projects which do not reveal the outcome of the contribution. These results are consistent with the hypothesis that unrestrictive developers are much more driven by commercial reputation motives than other groups. But the fact that public resolution appears to matter for other developer groups indicates that reputation in the form of peer recognition also plays some role.

[25] In total these projects receive 38,912 'closed' patches of which 8,945 do not report resolution, 24,855 report an 'Accepted' resolution and the remaining 5,112 projects report a 'Rejected' resolution.

[26] We also tried two alternative thresholds – 25 and 75 percent – to define the dummy variable for public resolution. While the parameter estimates depend on the threshold, the basic conclusion is robust.

Corporate Sponsorship of Projects: The second extension involves the role of corporate sponsorship of projects. Thus far we have inferred the role of reputation from the impact of receiving project size on the flow of contributions. But increasingly, large firms have invested substantial financial and technical resources to develop open source software innovation, including paying employees to participate in such projects.[27] It is highly likely that this investment involves corporate sponsorship of open source projects. Knowing which projects have such sponsorship should help pin down more sharply the role of the commerical repuation, as distinct from reputation associated with peer recognition. For developers motivated by commercial reputation, we would corporate sponsorship to increase the flow of contributions, conditional on the license type of the receiving project. We expect no such effect for anonymous or highly restrictive developers.[28]

Unfortunately, SourceForge does not identify whether the project has some form of commercial sponsorship. To examine this issue, we conducted an email survey of project managers of 500 projects listed on SourceForge to determine whether there was such sponsorship. We received answers from 222 projects, of which 45 percent reported corporate sponsorship. The distribution of projects licenses is similar for the survey respondents and the overall sample.[29] The survey projects are much larger than the average in the data set (mean numbers of patches received are 121 and 16.4, respectively), but within the survey, project size and the pattern of licenses are similar for corporate sponsored and individual projects.

Using this survey sample, we re-estimated the specification with linear dummies for receiving and contributing project characteristics (corresponding to Table 6) but including an interaction between developer type and corporate sponsorship. Table 9 presents the results. We get similar sorting patterns as in Table 6 for this much smaller sample – the magnitudes of the coefficients differ (as does the mean number of contributions) because of the focus on larger projects here. The key new finding is that corporate sponsorship has a powerful and highly significant effect on contributions by unrestrictive developers. The impact is nearly three times as large as the mean number of contributions for that group. There is no statistically significant effect for the other

[27] A discussion of IBM's involvement in open source software is available at http://www.research.ibm.com/journal/sj/442/capek.pdf

[28] In fact, the coefficient could be *negative* for these developers if their intrinsic motivation is associated with an assertive dislike for corporate sponsored software.

[29] In the SourceForge data, 58.5 percent of projects have highly restrictive licenses, and 18.6 percent are unrestrictive. The corresponding figures for the survey respondents are 49.5 and 21.2 percent.

developer groups. This finding strongy supports the importance of the commercial reputation motive for unrestrictive developers.

6.4. Evidence on Reciprocity

We begin with some summary statistics on the incidence of reciprocity among projects in open source development (Table 10).[30] We distinguish between two types of reciprocity. *Outward reciprocity* is the percentage of all the contributions made by project i which go to projects that previously contributed to project i. *Inward reciprocity* is the percentage of all contributions received by project i that come from projects to which project i had contributed previously. If all projects behaved symmetrically, outward and inward reciprocity would be the same. But we will see that they are very different.

Three features are striking. First, recpirocal contributions are very rare. Overall, only 1.37 percent of projects involve any reciprocal contributions.[31] Second, unrestrictive projects are much more likely to involve reciprocity (7.35 percent for unrestrictive versus 0.76 percent for highly restrictive projects). This is somewhat surprising since one might think that motivated agents would be more likely to reciprocate than commerically oriented developers. Third, among those projects involving some degree of reciprocal contributions, outward reciprocity plays a large role – accounting for 27 percent of all outgoing contributions for those projects, and again the degree of reciprocity is greatest for unrestrictive projects. Finally, outward reciprocity is much more frequent than inward reciprocity. This reveals that there is strong sorting by projects in terms of the degree to which they reciprocate for past contributions received.

These simple statistics may overstate the importance of genuine recirprocity because some two-way contributions may simply reflect similarities between projects. To examine this hypothesis, we performed probit regressions on the probability that a contribution is reciprocal against a set of dummy variables that capture whether the contributing and receiving projects

[30] We focus on reciprocity at the project level because we want to be able to control for matching between projects on various dimensions. We also examined reciprocity at the developer level and found that the frequency and patterns are similar to those reported in the text.

[31] While truncation may cause us to underestimate the occurenece of reciprocation somewhat – since SourceForge has only been operating since 1999 – the number is so low that we think truncation is very unlikely to reverse this finding.

match on license type, programming language, operating system and intended audience (Table 11).

We find that the only matching that signficantly affects reciprocity is matching on license type (column 1). Reciprocity is much stronger between projects with the same license – the implied marginal effect is 0.192, which is a 50 percent increase in the mean probability of reciprocity. When we include the size of the contributing and receiving projects, the matching coefficients are nearly identical (column 2). The basic story does not change when we allow for the matching coefficients to vary with license type (column 3) and with intended audience (column 4). While the estimated coefficients on the license matching dummies are not statistically significant individually, they are significant jointly ($p - value = .049$). It is interesting to note that the point estimates suggest that matching is a less important determinant of reciprocity for highly restrictive projects, which is consistent with the raw statistics in Table 10. We do not reject the joint hypothesis that there is positive (and equal) matching for restrictive and unrestrictive projects, but no matching for highly restrictive projects.($p - value < .001$). Column (4) presents the estimates under these restrictions. We also find evidence that reciprocity is stronger when both developers belong to end user projects.

In short, there is strong sorting of projects in terms of reciprocity. It plays a large role for a very small percentage of projects. But overall, this evidence shows that the reciprocity motive is not what sustains open source development. This finding does not support the strong claims about the importance of reciprocity, made first by Raymond (2001) and others since.

6.5. Implications of Developer Sorting Behavior

Our empirical findings show that the choice of license type and other project characteristics affect the pattern of contributions. For example, more restrictive licences increase contributions by highly restrictive developers but, at the same time, reduce those of other developer types. Thus project managers (or sponsors) face tradeoffs in choosing project characteristics. Lerner and Tirole (2002) emphasize the tradeoff between the number of contributions and proprietary control in choosing the license type. But our evidence shows that there is also another tradeoff – with sorting behavior, the license type can affect the number of incoming contributions itself.

To illustrate this, we begin with a simple computation to show how the restrictiveness

of the license affects the expected total number of contributions received by a project. Let $\Delta_{l \longrightarrow L}$ denote the change in the expected number of contributions received associated with a change in license type from l to L. Using the parameter estimates from Table 6, we get $\Delta_{HR \longrightarrow R} = -1.05$, $\Delta_{HR \longrightarrow UR} = -1.27$ and $\Delta_{R \longrightarrow UR} = -0.22$. At the aggregate level (pooling different project profiles), these calculations indicate that highly restrictive licenses maximize the expected number of contributions.[32] Of course this does not mean that such licenses are the preferred choice since they involve more limited 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 rate of innovation in open source software. Of course, a full analysis of this impact would require that we take into account the quality, as well as quantity, of contributions (we are currently doing this in a new research project).

Next, we show how the choice of license affects the number of contributions for projects with different intended audiences, and compare the predictions on which license maximizes contributions to the observed frequency of license types. To do this, we re-estimated the model in Table 6 separately for projects whose intended audience is end users and those aimed at developers. The results (not reported for brevity) show the same pattern of sorting behavior by developers on license type, but the responsiveness coefficients differ depending on the intended audience of the receiving project.[33] We use these estimated marginal effects to predict how the choice of license type affects total contributions for each project type.

For end user projects, we get $\Delta_{HR \longrightarrow R} = -3.92$, $\Delta_{HR \longrightarrow UR} = -4.54$ and $\Delta_{R \longrightarrow UR} = -0.62$. For developer tool projects, the computations yield $\Delta_{HR \longrightarrow R} = 0.40$, $\Delta_{HR \longrightarrow UR} = 3.44$ and $\Delta_{R \longrightarrow UR} = 3.04$. These calculations imply that highly restrictive licenses would maximize the flow of contributions for end user projects, but that unrestrictive licenses do so for developer tool projects. To check consistency with the data, we compute the ratio between the actual frequency and the one predicted by a random (multinomial) distribution. A ratio greater (resp.

[32] Using nine bi-monthly observations on 71 SourceForge projects (*not* at the developer level), Fershtman and Gandal (2007) find that the number of contributed lines of code is larger for projects with *less restrictive* licenses. They include both contributions by members and non-members of the project, and they do not consider the impact of heterogeneity of developers and sorting on contributions, which is the focus of our paper. These differences, and the fact that they use lines of code rather than the number of contributors, may account for the difference in findings.

[33] For end user projects, the results are very similar to those reported in Table 6. For developer oriented projects we find only weak sorting by anonymous and highly restrictive developers (though always in the direction of more restrictive licenses), but unrestrictive and mixed developers still strongly sort toward more restrictive licenses.

less) than one indicates over-representation (resp. under-representation) of projects with that license type. For end user projects, the ratio of actual to predicted frequency is 1.49 for HR, 0.36 for R and 0.47 for UR. For developer tool projects the ratios are 0.63 for HR, 1.44 for R and 1.29 for UR. As predicted, highly restrictive licenses are over-represented for end user projects, while the opposite holds for developer tools.[34]

7. Concluding Remarks

This paper studies the role of intrinsic motivation, reputation and reciprocity motives in driving open source software innovation. The empirical analysis is based on a large scale data set with detailed information about software code contributions to open source projects and the characteristics of the contributing and receiving projects. We study the pattern of contributions by four distinct developer groups – highly restrictive, unrestrictive, mixed, and anonymous – and infer the underlying motivations from the 'revealed preference' of developers. The central empirical finding is that developers of different types strongly sort on observed project characteristics – most notably, the restrictiveness of the license, project size, and corporate sponsorship. The empirical pattern of sorting behaviour points to an important role for motivated agents, reputation (especially commercial reputation, but also peer recognition), and to a lesser extent the reciprocity motive, in sustaining open source software innovation.

There are several directions for further research. The first is to develop and estimate an empirical framework that incorportes both the choice of license contract and the resulting sorting effects – integrating the work in this paper with Lerner and Tirole (2002). A second direction is to study how sorting affects the performance outcomes of projects – i.e., the quality of innovation. The recently redesigned website for SourceForge provides potentially useful information to study this link, such as the number of clicks and downloads for each project over time. Such measures could also be used to control for unobserved heterogeneity at the project level and thus more sharply identify the causal links underlying the sorting which we document in this paper. Finally, it would be useful to exploit any available information on changes in project license and corporate sponsorship to pin down the causal impact of these characteristics on contributions and innovation outcomes.

[34] We strongly reject the null hypothesis that contributions by license type are independent of intended audience ($p - value < .001$).

References

Ackerberg, Daniel and Maristella Botticini (2002), "Endogenous Matching and the Empirical Determinants of Contract Form," *Journal of Political Economy,* 110(3): 564-91

Benabou, Roland and Jean Tirole (2003), "Intrinsic and Extrinsic Motivation," *Review of Economics Studies*, 70(3): 489-520

Besley , Timothy and Maitreesh Ghatak (2005), "Competition and Incentives with Motivated Agents," *American Economic Review,* 95(3): 616-36

Bruns, Bryan (2001), "Open Sourcing Nanotechnology Research and Development: Issues and Opportunities," *Nanotechnology* (Institute of Physics), 12: 198-210

Delgaauw, Josse and David Perez-Castrillo (2004), "Incentives and Workers' Motivation in the Public Sector," Center for Economic Studies, CESinfo Working Paper 1223

Delfgaauw, Josse and Robert Dur (2008), "Incentives and Workers' Motivation in the Public Sector," *Economic Journal*, 118: 171-91

Delfgaauw, Josse and Robert Dur (2007), "Signalling and Screening of Workers' Motivation," *Journal of Economic Behavior and Organization*, 62(4): 605-24

Fershtman, Chaim and Neil Gandal (2007), "Open Source Software: Motivation and Restrictive Licensing," *International Economics and Economic Policy,* 4: 209-25

Francois, Patrick (2000), "Public Service Motivation as an Argument for Government Provision," *Journal of Public Economics,* 78(3): 275-99

Frey, Bruno (1997), *Not Just for the Money* (Cheltenham: Elgar Publishers)

Frey, Bruno and Felix Oberholzer-Gee (1997), "The Cost of Price Incentives: An Empirical Analysis of Motivation Crowding-Out," *American Economic Review*, 87(4): 746-55

Glazer, Amihai. (2004), "Motivating Devoted Workers," *International Journal of Industrial Organization,* 22(3): 427-40

Hann, Il-Horn, Jeff Roberts, Sandra Slaughter, and Roy Fielding (2004), "An Empirical Analysis of Economic Returns to Open Source Participation," Working paper, Carnegie-Mellon University

Haruvy, Ernan, Fang Wu, and Sujoy Chakravarty (2003), "Incentives for Developers' Contributions and Product Performance Metrics in Open Source Development: An Empirical Investigation," Working paper, University of Texas at Dallas

Hertel, G., M. Krishnan and Sandra Slaughter (2003), "Motivation in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel," *Research Policy,* 32(7): 1159-77

Johnson, Justin (2002), "Open Source Software: Private Provision of a Public Good," *Journal of Economics and Management Strategy*, 11: 637-62

Johnson, Justin (2004), "Collaboration, Peer Review and Open Source Software," Unpublished working paper, Cornell University

Kreps, David, "Intrinsic Motivation and Extrinsic Incentives," *American Economic Review Papers and Proceedings*, 87(2): 359-64

Lakhani, Karim, and Eric von Hippel (2003), "How Open Source Software Works: 'Free' User-to-User Assistance," *Research Policy*, 32: 923-43

Lerner, Josh, and Jean Tirole (2001), "The Open Source Movement: Key Research Questions," *European Economic Review,* 45(4-6): 819-26

Lerner, Josh, and Jean Tirole (2002), "Some Simple Economics of Open Source," *Journal of Industrial Economics,* 52: 197-234

Lerner, Josh and Jean Tirole (2005), "The Economics of Technology Sharing: Open Source and

Beyond," *Journal of Economic Perspectives,* 19(2): 99-120

Lerner, Josh, and Jean Tirole (2009), "The Scope of Open Source Licensing," *Journal of Law, Economics, and Organization,* 21: 20-56

Maurer, Stephen and Suzanne Scotchmer (2006), "Open Source Software: The New Intellectual Property Paradigm,"NBER Working Paper 12148

Murdock, Kevin (2002), "Intrinsic Motivation and Optimal Incentive Contracts," *RAND Journal of Economics,* 33(4): 650-71

Prendergast, Canice (2008), "Intrinsic Motivation and Incentives," *American Economic Review,* 98(2): 201-05

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

Trajtenberg, Manuel, Gil Shiff and Ran Melamed (2006), *"The 'Names Game': Harnessing Inventors' Patent Data for Economic Research,"* NBER Working Paper 12479

TABLE 1

CODE CONTRIBUTIONS AND RECEIVING PROJECT CHARACTERISTICS

	# Projects	# Patches Received	Mean	Std. Dev.	50th	90th
LICENSE TYPE						
Highly Restrictive	2,143	28,100	13.1	74.5	3	21
Restrictive	613	14,636	23.9	211.7	4	29
Unrestrictive	601	12,455	20.7	151.3	3	33
INTENDED AUDIENCE						
Developers	894	16,050	18.0	175.0	3	26
End-users/Desktop	606	9,635	15.9	123.4	3	20
Other	1,857	29,506	15.9	94.4	3	26
# OF MEMBERS						
Total	3,357	55,191	16.4	125.8	3	25
1	1,008	4,857	4.8	11.5	2	10
2 - 5	1487	11,514	7.7	16.5	3	17
6 - 10	502	10,258	20.4	43.6	6	52
11 - 50	337	21,821	64.8	329.4	9	106
Above 50	23	6,741	293.1	749.6	29	682
OPERATING SYSTEM						
Microsoft	225	3,555	15.8	69.7	2	16
OS Independent	989	12,956	13.1	42.9	3	24
POSIX	1,062	9,983	9.4	24.6	3	19
Multiple	1,081	28,647	26.5	213.9	3	35
PROGRAMMING LANGUAGE						
Total	3,357	55,191	16.4	125.8	3	25
Object Oriented	1,185	19,644	16.6	154.2	3	25
Imperative	1,138	20,609	18.1	141.2	3	25
Scriptive	449	7,184	16.0	47.9	3	30
Dynamic	463	5,276	11.4	34.2	3	24
Other	122	2,478	20.3	77.5	3	26

Notes: This table reports the number of contributions (patches) received by projects with different characteristics. The sample includes only projects that receive at least one contribution from developers that are not members of the receiving project. The project license defines the extent to which the developed code can be used for commercial purposes. Number of members is the number of developers who are closely affiliated with the project. Intended audience is the user group to whom the project is targeted. Operating system is the platform on which the program runs.

TABLE 2

CODE CONTRIBUTIONS AND CONTRIBUTING PROJECT CHARACTERISTICS

	# Projects	# Patches Submitted	Mean	Std. Dev.	50th	90th
LICENSE TYPE						
Highly Restrictive	4,368	16,218	3.7	7.5	2	8
Restrictive	1,162	6,914	6.0	22.3	2	10
Unrestrictive	1,261	6,474	5.1	9.4	2	12
INTENDED AUDIENCE						
Developers	1,826	8,491	5.4	18.5	2	11
End-users/Desktop	1,285	5,036	4.0	7.2	2	9
Other	3680	13,929	4.0	8.2	2	8
# OF MEMBERS						
Total	6,791	29,606	4.4	11.7	2	9
1	2,436	7,870	3.2	7.4	1	6
2 - 5	2,890	10,742	3.7	7.5	2	8
6 - 10	808	4,213	5.2	9.2	2	12
11 - 50	616	5,647	9.2	27.9	3	18
Above 50	41	1,134	27.7	37.0	12	88
OPERATING SYSTEM						
Microsoft	479	2,009	4.2	11.3	1	8
OS Independent	2,092	8,625	4.1	11.1	2	8
POSIX	2,097	7,384	3.5	5.8	2	8
Multiple	2,123	11,588	5.5	16.1	2	11
PROGRAMMING LANGUAGE						
Total	6,791	29,606	4.4	11.7	2	9
Object Oriented	2,604	11,829	4.5	15.7	2	8
Imperative	1,992	8,527	4.3	7.8	2	10
Scriptive	787	2,804	3.6	6.3	2	7
Dynamic	1,098	5,330	4.9	11.0	2	11
Other	310	1,116	3.6	6.1	2	7

Notes: This table reports the number of patches contributed by projects with different characteristics. The sample includes only projects that make at least one contribution. We exclude contributions by developers who are members of the receiving project. For definitions of other variables, see notes to Table 1.

TABLE 3

CODE CONTRIBUTIONS BY DEVELOPER TYPE

	# Developers	# Patches Contributed	Mean	Std. Dev.	50th	90th
All listed developers:	113,191	81,267	0.72	8.6	0	1
Anonymous developers:		5,699				
Origin of contribution:						
External Contributors:	13,060	53,194	4.1	51.0	1	6
Internal Contributions:	2,839	33,772	11.9	41.8	3	21
DEVELOPER TYPE						
Highly Restrictive	2,178	8,604	4.0	9.3	2	8
Mixed	2,501	19,446	7.8	21.8	3	15
Unrestrictive	524	1,991	3.8	6.7	2	8
Non-Members	7,856	17,454	2.2	5.2	1	4

Notes: This table reports the distribution of code contributions by developers of different types. We exclude contributions by developers who are members of the receiving project. 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 not belong to any project, but whose identity is known.

TABLE 4

DISRIBUTION OF CODE CONTRIBUTIONS BY DEVELOPER TYPE AND RECEIVING PROJECT CHARACTERISTICS, %

	Contributing developers				
	Anonymous	Highly Restrictive	Unrestrictive	Mixed	Non-members
Receiveing projects					
LICENSE TYPE					
Highly Restrictive	60.7	66.8	26.9	41.2	57.6
Restrictive	21.5	21.0	23.2	28.3	23.1
Unrestrictive	17.8	12.3	49.9	30.5	19.3
INTENDED AUDIENCE					
Developers	26.2	25.4	34.7	36.0	36.0
End users/Desktop	18.6	27.5	8.9	13.0	20.3
Other	55.2	47.1	56.4	51.0	83.8
# OF MEMBERS					
1	10.8	10.2	9.6	9.8	9.8
2 - 5	25.9	22.8	18.7	38.5	38.5
6 - 10	22.1	17.8	17.1	21.8	21.8
11 - 50	34.2	40.2	30.7	16.2	16.2
Above 50	7.0	9.0	23.9	13.6	13.6

Notes: This table reports the pattern of code contributions by different developers groups to projects with different characteristics. We exclude contributions by developers who are members of the receiving project. For definitions of other variables, see notes to Table 1.

Table 5

DEVELOPER TYPES AND THE LICENSE OF THE RECEIVING PROJECT

DEPENDENT VARIABLE: # OF CONTRIBUTIONS (4,770 OBSERVATIONS)

		Developer type			
		(1)	(2)	(3)	(4)
	Receiving project license type	Anonymous	Highly restrictive	Unrestrictive	Mixed
(a)	Highly restrictive	-1.07**	-	-3.08**	1.32**
		(0.244)		(0.157)	(0.436)
(b)	Restrictive	-1.44**	-0.858**	-2.29**	1.84**
		(0.216)	(0.136)	(0.149)	(0.579)
(c)	Unrestrictive	-1.62**	-1.38**	-2.22**	1.67**
		(0.192)	(0.135)	(0.144)	(0.562)
(c)	log(# of members)	1.95**	1.69**	2.53**	1.26**
		(0.197)	(0.180)	(0.267)	(0.150)
	Average # contributions	3.21	4.06	0.946	6.27
	Hypotheses tests:				
	(a)=(c)	χ^2=18.33**	χ^2=56.52**	χ^2=19.04**	χ^2=1.13
	(a)=(b)=(c)	χ^2=22.25**	χ^2=69.42**	χ^2=26.42**	χ^2=3.93

Notes: This table reports the estimated marginal effects (evaluated at the mean) of the interaction terms between the contributing developer type and both the license type and number of members of the receiving project. The regression also 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.

The model is estimated by Poisson QMLE. Estimated standard errors are in parentheses. * denotes statistical significance at the 5% level, and ** at the 1% level.

Table 6

DEVELOPER TYPES AND THE LICENSE OF THE RECEIVING PROJECT, ADDING INTENDED AUDIENCE INTERACTIONS

DEPENDENT VARIABLE: # OF CONTRIBUTIONS (4,770 OBSERVATIONS)

		Developer type			
		(1)	(2)	(3)	(4)
	Receiving projects	Anonymous	Highly restrictive	Unrestrictive	Mixed
(a)	Highly restrictive	-0.259	-	-3.52**	1.19**
		(0.924)		(0.489)	(0.174)
(b)	Restrictive	-0.689	-0.627**	-2.71**	0.389
		(0.769)	(0.147)	(0.272)	(1.516)
(c)	Unrestrictive	-0.954	-1.16**	-2.60**	0.852
		(0.662)	(0.145)	(0.239)	(1.689)
(c)	log(# of members)	1.91**	1.57**	2.62**	0.054
		(0.189)	(0.176)	(0.272)	(0.306)
(d)	Developers	14.56**	12.73**	20.19**	4.17**
		(3.51)	(3.08)	(5.16)	(0.699)
(e)	End user/Desktop	20.07**	18.55**	14.91**	2.99**
		(4.70)	(4.28)	(4.61)	(0.613)
	Average # contributions	3.21	4.06	0.946	6.27
	Hypotheses tests:				
	(a)=(c)	χ^2=12.06**	χ^2=38.19**	χ^2=7.62**	χ^2=0.07
	(a)=(b)=(c)	χ^2=13.96**	χ^2=42.30**	χ^2=9.96**	χ^2=0.74
	(d)=(e)	χ^2=105.22**	χ^2=104.77**	χ^2=88.64**	χ^2=96.30**

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, and two intended audience categories (developer tools and end-users) 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.

The model is estimated by Poisson QMLE. Estimated standard errors are in parentheses. * denotes statistical significance at the 5% level, and ** at the 1% level.

Table 7

DEVELOPER TYPES AND THE LICENSE OF THE RECEIVING PROJECT, ADDING CONTRIBUTING PROJECT CHARACTERISTICS

DEPENDENT VARIABLE: # OF CONTRIBUTIONS (32,710 OBSERVATIONS)

		Contributing		
		(1)	(2)	(3)
	Receiving	Highly restrictive	Unrestrictive	Mixed
(a)	Highly restrictive	-	-0.368**	0.208**
			(0.015)	(0.035)
(b)	Restrictive	-0.087**	-0.265**	0.163**
		(0.009)	(0.013)	(0.038)
(c)	Unrestrictive	-0.145**	-0.238**	0.242**
		(0.009)	(0.014)	(0.044)
(c)	log(# of members)	0.274**	0.323**	0.219**
		(0.015)	(0.021)	(0.016)
	Average # contributions	0.499	0.732	0.122
	Hypotheses tests:			
	(a)=(c)	χ^2=153.9**	χ^2=111.9**	χ^2=56.4**
	(a)=(b)=(c)	χ^2=167.2**	χ^2=115.6**	χ^2=82.9**

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, and two intended audience categories (developer tools and end users) 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, *and* a complete set of linear dummies for the contributing project intended audience and programming language. We also include a linear control for the number of projects in the cell.

The model is estimated by Poisson QMLE. Estimated standard errors are in parentheses. * denotes statistical significance at the 5% level, and ** at the 1% level.

Table 8

DEVELOPER TYPES AND PUBLIC RESOLUTION

DEPENDENT VARIABLE: # OF CONTRIBUTIONS (3,265 OBSERVATIONS)

		Developer type			
		(1)	(2)	(3)	(4)
	Receiving projects	Anonymous	Highly restrictive	Unrestrictive	Mixed
(a)	Highly restrictive	-1.121**	-	-4.135**	1.999**
		(0.439)		(0.240)	(0.742)
(b)	Restrictive	-1.758**	-1.019**	-3.235**	1.892**
		(0.364)	(0.231)	(0.203)	(0.844)
(c)	Unrestrictive	-1.527**	-1.421**	-2.508**	2.884**
		(0.391)	(0.246)	(0.289)	(0.999)
(d)	log(# of members)	1.810**	1.552**	2.149**	1.199**
		(0.202)	(0.191)	(0.295)	(0.158)
(e)	Dummy for public resolution	0.909**	1.048**	2.771**	1.047**
		(0.396)	(0.372)	(0.994)	(0.308)
	Average # contributions	4.016	4.860	1.387	7.698
	Hypotheses tests:				
	(a)=(c)	χ^2=3.04	χ^2=20.30**	χ^2=51.25**	χ^2=2.50
	(a)=(b)=(c)	χ^2=10.44**	χ^2=27.31**	χ^2=53.30**	χ^2=3.58

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. The public resolution dummy 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.

Table 9

CORPORATE SPONSORSHIP

DEPENDENT VARIABLE: # OF CONTRIBUTIONS (728 OBSERVATIONS)

		Developer type			
		(1)	(2)	(3)	(4)
	Receiving project license type	Anonymous	Highly restrictive	Unrestrictive	Mixed
(a)	Highly restrictive	-8.046*	-	-14.769**	-6.287*
		(3.530)		(2.062)	(3.233)
(b)	Restrictive	-10.648**	-7.606**	-13.214**	-9.288**
		(2.312)	(1.612)	(1.685)	(2.217)
(c)	Unrestrictive	-9.999**	-8.006**	-10.774**	-2.550
		(2.234)	(1.678)	(1.964)	(4.535)
(d)	Dummy for corporate sponsorship	4.390	-0.522	19.211*	4.384
		(4.893)	(2.794)	(9.582)	(2.643)
	Average # contributions	12.61	24.5	7.33	47.66
	Hypotheses tests:				
	(a)=(c)	χ^2=2.31	χ^2=9.37**	χ^2=2.67	χ^2=4.97*
	(a)=(b)=(c)	χ^2=4.62	χ^2=15.66**	χ^2=5.32	χ^2=27.38**

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 is corporate-sponsored. The sample used for this regression is based on a sample of 222 projects in SourceForge, based on an email survey we sent to 500 of the larger projects to determine whether they were corporate-sponsored. 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.

The model is estimated by Poisson QMLE. Estimated standard errors are in parentheses.
* denotes statistical significance at the 5% level, and ** at the 1% level.

Table 10

PATTERN OF RECIPROCITY

Receiving license type:	# of projects	Total patches received	Total patches contributed	% outward patch reciprocity	% inward patch reciprocity
All	83	11,442	2,703	26.97	5.22
Highly Restrictive	29	3,434	426	27.23	3.17
Restrictive	19	3,806	1,120	13.57	1.42
Unrestrictive	35	4,202	1,157	39.84	10.33

Notes: This table reports the pattern of reciprocity of contributions for projects with different license types. Outward reciprocity is defined as the percentage of all the contributions made by project i which go to projects that previously contributed to project i. Inward reciprocity is the percentage of all contributions received by project i that come from projects to which project i had contributed previously.

Table 11

DETERMINANTS OF RECIPROCITY

DEPENDENT VARIABLE: DUMMY FOR RECIPROCITY (1,964 OBSERVATIONS)

	(1)	(2)	(3)	(4)	(5)
Dummy for matching on license type	0.192** (0.070)	0.183** (0.067)			0.180** (0.067)
Dummy for matching on intended audience	0.057 (0.059)	0.050 (0.053)	0.057 (0.053)	0.060 (0.054)	
Dummy for matching on programming language	-0.046 (0.055)	-0.029 (0.047)	-0.028 (0.048)	-0.026 (0.048)	-0.044 (0.045)
Dummy for matching on operating systems	0.032 (0.054)	0.011 (0.058)	0.007 (0.059)	0.008 (0.059)	-0.001 (0.058)
Rceieving size		0.004** (0.002)	0.004** (0.002)	0.004** (0.002)	0.004** (0.001)
Contributing size		0.003 (0.002)	0.003 (0.002)	0.003 (0.002)	0.003 (0.002)
Dummy for matching on Highly restrictive			0.076 (0.131)		
Dummy for matching on Restrictive			0.261 (0.177)		
Dummy for matching on Unrestrictive			0.234 (0.162)		
Dummy for matching on Restrictive or Unrestrictive				0.253** (0.098)	
Dummy for matching on Developers					0.152 (0.189)
Dummy for matching on End User					0.634** (0.057)
Dummy for matching on Multiple					-0.126 (0.155)
Average reciprocity:	0.371	0.371	0.371	0.371	0.371
R^2	0.036	0.109	0.110	0.110	0.118

Notes: This table reports estimates from (Probit) regressions for whether a code contribution is reciprocal. The dummy equals one if the contribution is reciprocal and zero otherwise. A patch from project *i* to project *j* made in year *t* is defined as reciprocal if project *i* received a contribution from *j* prior to year *t.* 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. All regressions also include complete sets of dummies for receiving license type, intended audience, programming language and operating systems.

Standard errors (in brackets) 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 A.1

CLASSIFICATION OF LICENSE TYPES

License Name	Highly Restrictive	Restrictive	Unrestrictive	# of projects
Academic Free License			X	240
Apache License			X	1,451
Apple Public Source License		X		50
Artistic License			X	1,175
Attribution Assurance License			X	25
BSD License			X	4,550
Common Development and Distribution License		X		22
Common Public License		X		512
CUA Office Public License Version 1.0		X		2
Eclipse Public License			X	78
Educational Community License		X		8
Eiffel Forum License		X		23
Fair License			X	9
GNU General Public License	X			42,234
GNU Library or Lesser General Public License		X		7,208
Historical Permission Notice and Disclaimer			X	13
IBM Public License		X		77
Intel Open Source License			X	20
Jabber Open Source License		X		38
MIT License			X	1,156
MITRE Collaborative Virtual Workspace License		X		2
Motosoto License		X		1
Mozilla Public License		X		974
Nethack General Public License		X		31
Nokia Open Source License		X		8
Open Group Test Suite License			X	16
Open Software License		X		275
PHP License		X		144
Python License			X	268
Qt Public License		X		211
Reciprocal Public License		X		19
Ricoh Source Code Public License		X		8
Sleepycat License		X		16
Sun Industry Standards Source License			X	41
Sun Public License		X		49
University of Illinois/NCSA Open Source License			X	35
Vovida Software License 1.0			X	4
W3C License			X	34
wxWindows Library Licence		X		50
X.Net License			X	7
zlib/libpng License			X	323
Zope Public License			X	27
Public Domain				1,525

TABLE A.2

PANEL A: MOST ACTIVE CONTRIBUTING PROJECTS

Name	Topic	License	License Type	Age	# Developers	# Patches received	# Patches Contributed
Python	Interpreters	Python	Unrestrictive	6	63	9,483	1,515
WinMerge	Version Control	GNU GPL	Highly Restrictive	6	10	1,884	1,066
wxWidgets	Frameworks	wxWindows Library	Restrictive	6	27	5,126	538
Tcl	Interpreters	BSD	Unrestrictive	6	46	2,110	523
Tk Toolkit	Desktop Environment	BSD	Unrestrictive	6	42	795	507
SpamBayes	Filters	Python Software	Unrestrictive	6	25	160	490
BZFlag	Simulation	GNU Lesser GPL	Restrictive	6	62	976	406
ht2html	Site Management	Python	Unrestrictive	6	8	15	399
ScummVM	Games/Entertainment	GNU GPL	Highly Restrictive	5	35	1,010	373
wxCode	Software Development	wxWindows Library	Restrictive	6	22	7	361

PANEL B: MOST ACTIVE RECEIVING PROJECTS

Name	Topic	License	License Type	Age	# Developers	# Patches received	# Patches Contributed
Python	Interpreters	Python	Unrestrictive	6	63	9,483	1,515
wxWidgets	Frameworks	wxWindows Library	Restrictive	6	27	5,126	538
Gaim	Internet Relay Chat	GNU GPL	Highly Restrictive	7	14	3,718	318
Tcl	Interpreters	BSD	Unrestrictive	6	46	2,110	523
WinMerge	Version Control	GNU GPL	Highly Restrictive	6	10	1,884	1,066
ScummVM	Games/Entertainment	GNU GPL	Highly Restrictive	5	35	1,010	373
BZFlag	Simulation	GNU Lesser GPL	Restrictive	6	62	976	406
OpenTTD	Simulation	GNU GPL	Highly Restrictive	2	11	874	182
net-snmp	Internet	BSD	Unrestrictive	6	15	870	178
SCons	Build Tools	MIT	Unrestrictive	5	15	838	192

EDS Innovation Research Programme
London School of Economics & Political Science
Lionel Robbins Building
Houghton Street
London WC2A 2AE
020 7955 7285

www.lse.ac.uk/eds