Network Dynamics, Knowledge Transfer, and Incremental Innovations in Virtual Organizations[1]

Neil Gandal
Berglas School of Economics
Tel Aviv University
Ramat Aviv, Tel Aviv 69978, Israel
Email: gandal@post.tau.ac.il

Uriel Stettner
Recanati Business School
Tel Aviv University
Ramat Aviv, Tel Aviv 69978, Israel
Email: urielste@tau.ac.il

June 21, 2015

Abstract:

Employing a model of knowledge spillovers, we find empirical evidence consistent with both direct and indirect spillovers among open source software projects. This result is robust to analyses using a fixed effect model or a difference model looking at differences over the 2006 to 2009 period. There is also evidence that the addition of a "star" programmer to the project is associated with greater success, even after controlling for the induced changes in the network structure that are associated with the star's additional direct connections to other projects. Interestingly, we further find strong evidence that project modifications are positively associated with project success. This suggests that "incremental" innovations are critical to project success.

Keywords: Network Dynamics, Knowledge Spillovers, Modification of Code, Social Network, Open Source

[1] We appreciate the invaluable research assistance of Yaniv Friedensohn and Peter Naftaliev and gratefully acknowledge the financial support of the Israel Science Foundation (grant No. 1287/12) and the support of a grant by the Research Program for the Economics of Knowledge Contribution and Distribution. We are grateful to seminar participants at UT-Austin and the Economics of Knowledge Contribution and Distribution conference at Georgia Institute of Technology for helpful comments. Any opinions expressed are those of the authors.

1. Introduction

Knowledge spillovers enable firms and researchers to benefit from innovations of others. Software engineering is a particular setting in which such knowledge spillovers are likely to be important in product development given the rapid change in development methodologies, changing product-market preferences, and increasing competitive pressures. Open source software (OSS) can facilitate spillovers in R&D because such "source" code is available to the broad public under a variety of OSS licenses (Laurent, 2004). Such licenses grant the rights to use the entire work, to create a derivative work, or to share or market such work subject to the license governing the specific open-source project (Bonaccorsi, Rossi, & Giannangeli, 2006; Von Hippel & Von Krogh, 2003; Lerner & Tirole, 2002). Accordingly, one of the central aspects of OSS development is the ability to share and absorb knowledge that has been created outside of a distinct OSS project. Such spillovers facilitate the transfer of knowledge and ideas among OSS projects. In its traditional form, open-source software (OSS) development is a collaborative effort of loosely coordinated and geographically dispersed developers who contribute their time and knowledge to establishing and improving software and whose underlying knowledge is made accessible to the general population. OSS projects, like virtual teams, are semi-structured groups of skilled developers working on interdependent tasks using informal, non-hierarchical, and decentralized communication with the common goal of creating a valuable product (Lipnack & Stamps, 1997).

Knowledge spillovers across projects often occur via contributors who for example adapt software so that it can function in a computing environment that is different from the one for which it was originally designed (i.e., porting), provide alternative value propositions on the basis of previously existing software solutions (i.e., forking), or apply distinct pieces of

functionality to different development efforts settings in attempt to benefit from previous development efforts (i.e., reusing). Direct spillovers occur when projects have a common developer who transfers information and knowledge (primarily code) from one project to another. In contrast, indirect project spillovers occur when knowledge is transferred from one project to another when the two projects are not directly linked i.e., there is no common contributor. For example, suppose that programmer "A" works on projects I and II, while programmer "B" works on projects II and III. Programmer A could take innovative source code from project I and use it in project II. Programmer B might find that source code useful – and port, fork, or reuse it over from project II to project III. In such a case, knowledge is transferred from one project to another by programmers who work on more than one project. There is a direct spillover from project I to project II, and an indirect spillover from project I to project III, since projects I and III are not directly connected.

Sourceforge.net facilitates software developer collaboration by providing a free online platform for managing projects, communications, and software code. It is the largest repository of registered OSS development projects during the period of our study, with tens of thousands of projects and contributors. In addition to providing information about the project (date established, project stage, etc.), each Sourceforge.net project contains a list of registered team members who contribute their time and knowledge to the advancement of one or more OSS projects.

Following Grewal, Lilien, & Mallapragada (2006) and Fershtman and Gandal (2011), we construct a *project network* by defining two projects to be connected if they have a developer in common. In the example above, there is a direct link between projects I and II and a direct link between projects II and III. We can also construct the related *contributor network* by defining

two contributors as connected if they work together on the same project. In the example above, there is a direct link between contributors A and B, since they both work on project II. In addition to constructing the project and contributor networks, we have painstakingly calculated the number of *modifications* and *additions* made to the code for each project over the study period. A modification is a change made by a single programmer to existing code, while an addition is the case in which a developer adds a new block of code. Thus, a modification captures an activity on an existing file that contains a particular set of software instructions (code). This could include changes such as making the code more efficient by reducing the lines of code or adding new lines of code making the code more stable. In contrast, an addition captures the introduction of a new file that contains code a particular set of software instructions (code) that was not previously part of a focal OSS project. Thus, our unit of measurement is a "file" containing one or more lines of software code. We extracted information on addition/modification activities (i.e., "commits") of individual contributors to the repository of a particular project in a particular year.

Modifications are a good proxy for incremental innovation that, for example, improve how the software product works, while additions are a proxy for new, additional functionality, and other changes in product capabilities. We find that in general, changes in the network architecture are positively associated with changes in project success. Employing a model of knowledge spillovers, we find evidence that is consistent with both direct and indirect spillovers. This result is robust to running the analysis using a fixed effect model or a difference model, i.e., looking at differences over the 2006 to 2009 period. There is also evidence that the addition of a "star" programmer on the project is associated with greater success, even after controlling for the

induced change in the network structure that is associated with the star's additional direct connections to other projects.

Interestingly, we further find that modifications are significantly associated with project success and this result is extremely robust. This suggests that "incremental" innovations are critical to project success. On the other hand, the results regarding the relationship between additions and project success are more ambiguous; in some specifications, additions of software code are associated significantly with project success, while in other specifications they are not significantly associated with project success. The importance of modifications rather than additions is a very important result. It casts some doubt on a firmly held belief, namely that the size of the code (lines of codes, modules) is a good proxy itself for success in open source projects.

Several other recent studies have examined the relationship between network structure and performance (Ahuja, 2000; Calvó-Armengol, Patacchini, & Zenou, 2009).[2] Our paper is closest to that of Fershtman and Gandal (2011), who focus on spillovers that occur by means of the interactions of different researchers or developers in OSS projects. The theoretical model developed by Fershtman and Gandal (2011) shows that project spillovers imply (i) positive association between degree and project success and (ii) positive associations between closeness and project success. Using cross-sectional data, they demonstrate that the structure of the product network is associated with the project's success, which provides support for knowledge spillovers.

[2] Other recent studies have examined the relationship between network structure and behavior (e.g., Ballester, Calvó-Armengol, & Zenou, 2006; Calvo-Armengol & Jackson, 2004; Goyal, van der Leij and Moraga-Gonzalez (2006); Jackson & Yariv, 2007; Karlan, Mobius, Rosenblat, & Szeidl, 2009).

None of these papers examines the relationship between changes in the network architecture and changes in success over time. Perhaps more importantly, this is the first paper that measures input to open source projects at the micro-micro level, by taking account of every contribution to the software code, both via modifications of existing code, or by additions of new blocks of code.

2. Research Setting and Data

This paper uses a replica of publicly available data from Sourceforge.net that is hosted at Notre Dame University. Sourceforge.net facilitates software developer collaboration by providing a free online platform for managing projects, communications, and software code. Sourceforge.net is the largest repository of registered OSS development projects during the period of our study. Each SourceForge.net project contains a list of registered team members who contribute their time and knowledge to the advancement of an OSS project. Each project links to a "developer page" that contains meta-information on a particular contributor, including the date the developer joined the project, the developer's functional description (e.g., administrator, developer) and his or her geographic location. Because accessibility to OSS projects is unrestricted and because the contributors can be identified by their unique user names, we utilize this information to construct a two-mode network that relates projects via registered contributors. Accordingly, we define two OSS projects as being connected when there are common contributors who participate in both projects.

Each project links to a standardized "Project page" that lists descriptive information on a particular project, including a statement of purpose, the intended audience, the license, and the operating system for which the application is designed. Moreover, a "Statistics page" shows various project activity measures, including the number of project page views and downloads

registered for the focal project. Although some data are available for other periods, statistics on downloads are available only for the 2006–2009 period. Therefore, we deploy yearly panel data from 2006–2009 to construct two distinct two-mode networks: (i) the project network and (ii) the contributor network. In the former, the nodes are the OSS projects, and two projects are linked when there are common contributors who work on both. In the latter, the nodes of the contributor network are the contributors, and two contributors are linked if they participated in at least one OSS project together.

Regarding the project network in 2009, we find that 84.3% percent of the projects have either one or two contributors, 9.2% have three to four contributors and 6.5% have five or more contributors (see Table 1). With regard to the contributor network in January 2009, 91.3% of the contributors worked on one or two projects, 6.5% of the contributors worked on three to four projects, and 2.1% of the contributors worked on five or more projects.[3] While we focus on the project network, our analysis also includes a key feature of the contributor network: contributors who work on five or more projects. We define such contributors as 'Stars.'

[3] These percentages were virtually identical in 2006 as well.

Table 1: Distribution of components in project networks—January 2009

Project Network		Contributor Network	
Contributors per project	*Percent of total projects*	*Projects per contributor*	*Percent of Contributors*
1	69.9	1	77.2
2	14.4	2	14.1
3-4	9.2	3-4	6.5
5-9	4.8	5-9	1.9
10 or more	1.7	10 or more	0.2

Having panel data from 2006 to 2009 is helpful because it is difficult to determine causality from cross-sectional data: this is because unobserved fixed project effects might be driving success. Because we do not have data on these fixed project effects, they are included in the error term when running cross-sectional analyses. If these unobserved effects are correlated with the right-hand-side variables, the estimates from the cross-sectional analysis will be biased; however, this problem is eliminated when using fixed effect models or estimating difference models, both of which we do in the paper.

2.1 Theory

While do not directly observe spillovers, we will employ a simple model from Fershtman and Gandal (FG 2011) in which spillovers can be proxied by two network centrality measures: (i) a project's degree, which is the number of projects with which the focal project has a direct link or common developers and (ii) a project's closeness centrality, which is the inverse of the sum of all distances between a focal project and all other projects multiplied by the number of

other projects. Intuitively, closeness centrality measures how far each project is from all the other projects in a network.[4] Closeness centrality is calculated as:[5]

$$(1) \qquad C_i \equiv \frac{(N-1)}{\sum_{j \in N} d(i,j)},$$

where $d(i,j)$ is the distance between project i and j. We start by assuming that the expected success level of each project "i" (without any spillovers) is given by

$$(2) \qquad S_{it} = \alpha + A_i'\delta + X_{it}\omega + \varepsilon_{it},$$

where S_{it} is the success of project i at time t, α is a constant, A_i is a vector of unobserved time-invariant project factors, X_{it} is a vector of observable time-varying factors , and ε_{it} is an error term. We now add the FG (2011) assumptions:

- Each project (potentially) receives a positive spillover denoted β from all 'connected' projects.
- The project also (potentially) enjoys positive spillovers from projects that are indirectly connected, but that these spillovers are subject to decay: the greater the distance between the projects in the projects network, the smaller the indirect spillovers. When the distance between project i and j is $d(i,j)$, this spillover is $\gamma / \sum_j d(i,j)$

Under these assumptions, the success level of each project i at time t can be written

$$(3) \qquad S_{it} = \alpha + A_i'\delta + X_{it}\omega + \beta D_{it} + \sum_j \gamma / d(i,j)_t$$

where D_{it} is the *degree* of project i in the network at time t, and β and γ are greater than or equal to zero.

[4] Closeness centrality lies in the range [0,1]. In the case of a Star network with a single project in the middle that is connected to all other projects, the closeness centrality of the project in the center is one.
[5] See Freeman (1979), pp. 225-226 and Wasserman and Faust (1994), pp. 184-185.

Using (1), the expression for closeness centrality, project i's success at time t can be rewritten as

$$(4) \qquad S_{it} = \alpha + \mathrm{A_i}'\delta + \mathrm{X_{it}}\omega + \beta D_{it} + \gamma C_{it}/[N-1]_i,$$

This spillover specification is simple but quite general. When β and γ equal zero, there are no spillovers at all. When β>0 and γ=0, there are only direct spillovers. When β=0 and γ>0, there are both direct and indirect spillovers which are exclusively measured by the projects closeness centrality. When β>0 and γ>0, there are additional spillovers from directly connected projects above and beyond those captured by its *closeness* measure: the spillovers have a 'hyperbolic' structure.

As is typical, we will estimate (4) using a fixed effects model. In the case of a fixed effects model, $\alpha_i \equiv \alpha + \mathrm{A_i}'\delta$ is a parameter to be estimated.[6]

2.2 Dependent Variable

Consistent with prior research, we measure project performance by examining the number of times a project has been downloaded (Fershtman & Gandal, 2011; Grewal et al., 2006). We focus on downloads of the executable, compiled product because users will not typically download the source code, where ldownloads ≡ ln(1+downloads,) where "ln" means the natural logarithm. In the case of software, downloading takes time and effort. Hence, downloads are considered to be an excellent proxy to success.

[6] As Angrist and Pischke (2009) note, treating α_i as a parameter to be estimated is equivalent to estimating in deviations from means. Fixed effects are also equivalent to estimating in differences if there are only two periods of data.

2.3 Independent Variables

We will use a log-log specification; hence the independent variables will be in log form as well. For the empirical analysis, we will use the following project network variables:

- The variable ldegree = ln(1+degree),
- The variable lcloseness = ln(0.05+closeness),[7]

  where project degree and project closeness were defined earlier. Other key variables of interest include the following:

- We define the dummy variable Star5 as follows. It takes on the value one if the contributor was a member of five or more projects. This variable comes from the contributor network, not the project network. Clearly, having a "Star" contributor join a project gives that project more connections to other projects. An interesting question is whether adding a "Star" to the team of developers has an effect on the success of a project beyond the effect it has on connectivity (i.e., network structure.)
- We have data on the number of modifications and additions for each project in each year. We define lmod=ln(1+number of modifications on the project.) We define ladd=ln(1+ number of additions to the project). Modifications and additions are measured like downloads. Hence, in 2009, the total number of modifications (additions) for each project is the sum of the modifications (additions) made during the 2006-2009 period.

[7] The reason we add such a small number is because the mean value of *closeness* is quite small.

In addition to the variables of interest we described above, we have data for a group of control variables:

- The variable years_since is the number of years that have elapsed since the project first appeared at Sourceforge: lyears_since=ln(years_since).
- The variable cpp is the number of contributors that participated in the project: lcpp=ln(cpp)

  The data from Sourceforge.net include information on the six possible stages of development for each product. The stages are:

  1 – Planning, 2 - Pre-Alpha, 3 – Alpha, 4 – Beta, 5 – Production/Stable, 6 – Mature.

  We denote the stage of development of the product as stage, where stage ranges from one to six.[8]

2.3 Discussion of the Data

Like most 'empirical' networks, the project network of OSS projects at SourceForge.net has one giant component and many small components. Closeness centrality is only defined for connected projects; hence we will primarily focus on the giant component. Nevertheless, we will also analyze projects outside the giant component as well. In such a case, we cannot include closeness in the analysis.

Descriptive statistics are shown in Table 4 in the appendix. Table 4 shows that projects in the giant component have on average many more downloads than projects outside of the giant

[8] A few of the projects have multiple stages listed. We exclude these projects from the analysis. Including these projects and taking the average stage as the stage of the project has no effect on the results.

component (96,998 vs. 18,839). Further, projects in the giant component have (i) more contributors (4.07 vs. 1.63), (ii) a larger *degree* (6.26 vs. 1.18), and a great number of stars (0.50 vs 0.08).[9]

3. Empirical Analysis:

The relationship between the number of contributors and *downloads* is likely non-linear: additional contributors are likely associated with a larger number of downloads, but the marginal effect of each additional contributor declines as the number of contributors increases. The same is likely true for the relationship between network variables and downloads as well, which suggests that a "log/log" model is appropriate. Thus, we use the following estimating equation: [10]

$$(5) \quad ldownloads = \beta_0 + \beta_1 *lcpp + \beta_2 * ldegree + \beta_3 * lcloseness + \beta_4*Star5 + \beta_5 *Stage + \beta_6 *lyears\_since + \beta_7 *ladd+ \beta_8 *lmod +\varepsilon$$

We estimate (5) for 2006-2009, the period for which we have data on downloads. The main results are shown in Table 2. Columns 1-4 in that Table are for projects outside of the giant component, while columns 5-8 are for projects inside the giant component.

In columns 1 and 5, we include all observations, but do not include fixed effects, while in columns 2 and 6 we include again all observations and also add fixed effects.

[9] Correlations among the network centrality variables in the giant component are shown in Table 5 in the appendix.
[10] All independent variables (except Stars and Stage) are in logarithmic form. As noted above, we denote this by including an 'l' before the variable name.

Network Dynamics and Knowledge Transfer

3.1 Direct and Indirect Project Spillovers

Columns 1 and 5 (outside and inside of giant respectively) in Table 2 show that degree centrality is positively associated with the number of downloads for projects both outside of and inside of the giant component – and that this association is statistically significant. This result continues to hold in columns 2 and 6 when we add the fixed effects.

Columns 5 and 6 in Table 2 also show that changes in closeness centrality are positively associated with downloads for projects in the giant component.[11] Since the coefficients on degree and closeness are both positive for projects in the giant component, this suggests that there are both direct and indirect project spillovers and that the spillovers have a hyperbolic structure, so that the direct spillover is quite large, relative to the indirect spillovers.

3.2 The Effect of Stars

Table 2 shows that without the fixed effects, the number of Stars is not positively associated with success. Once we include the fixed effects, however, we indeed find that the number of stars is positively associated with success both outside the giant component. Given that fixed effects models eliminate the bias associated with unobserved project effects, this suggests that changes in the number of Stars are positively associated with changes in the number of downloads even after controlling for the network structure.

3.3 Additions and Modifications

From the analysis with and without fixed effects (columns 1, 2, 5, and 6,) additions and modifications are positively associated with success, both inside and outside of the giant

[11] Recall that when we employ closeness in the analysis, we must restrict attention to connected projects, i.e., projects in the giant component.

component. In most of these regressions, however, the effect of modifications is stronger than the effect of additions. This result suggests that incremental changes may be more important for success than large changes and additional functionality.

4. Robustness Analysis:

4.1 Testing for Endogeneity

Although our analysis focuses on how the network structure affects success, the reverse may be true as well: contributors may want to join popular projects. Developers may want to be associated with very successful projects, thereby making the number of contributors (and thus the degree) endogenous. In fact, the Sourceforge.net website states that, "as a project's activity rises, SourceForge.net's internal ranking system makes it more visible to other developers who may join and contribute to it. Given that many open-source projects fail due to a lack of developer support, exposure to such a large community of developers can continually breathe new life into a project."

With the exception of Calvo-Armengol, Patacchini, and Zenou (CPZ, 2008), we are not aware of any empirical papers in the social network literature that estimate a structural model and (hence) are able to econometrically deal with the endogeneity issue by using instruments. Even if we could develop a structural model, it would likely depend on variables like effort or marginal cost that are not observable.

Hence, we must address the potential endogeneity of degree and closeness in another way. Not surprisingly, the distribution of changes in downloads is quite skewed. In the case of projects outside of the giant component, the mean yearly change in downloads is 7,558, while the 75^{th} percentile is 509. In the case of projects inside the giant component, the mean yearly change in downloads is 46,211, while the 75^{th} percentile is 1707. Although it is not ideal, we believe that

the best way to address this issue is to exclude the relatively few projects that had a lot of changes in downloads over time. The 'joining popular projects' effect is likely to be less of a factor for projects that did not experience significant increases in downloads (i.e., success).

In columns 3 and 7, we test for endogeneity by restricting ourselves to cases where the change in the number of downloads from year to year is relatively small. Hence, we re-run the analysis including projects with less than 509 and 1707 yearly additional downloads for projects (outside and inside of the giant component respectively.) When running this analysis, we lose the observations from 2006.[12]

As expected, in columns 3 and 7 in Table 2, the estimated coefficients on degree and the number of contributors (cpp) are much smaller, both for projects inside and outside of the giant component, suggesting that both cpp and degree are indeed potentially endogenous. Nevertheless, the estimated coefficient on degree is still statistically significant both inside and outside of the giant component. Similarly, the estimated coefficient on cpp is also significant both inside and outside of the giant component.

The estimated coefficient on closeness, on the other hand, does not fall. These results suggest that closeness is not endogenous. This makes sense, since closeness can only be endogenous under a very unlikely scenario.[13]

[12] It can be argued that it is better to exclude 2006 from the analysis in columns 1, 2, 5 and 6 as well. But since our results are robust to the inclusion of 2006 data, we include these data.

[13] The unlikely scenario is as follows: developers may want to work on a particular project so that a developer on that project can "introduce" them to a developer (on another project) whom they would like to meet. Since our network is a fairly thin one (many projects and relatively few developers) and that the average project in our dataset has less than four contributors to a project in the giant component, it is unlikely that this indirect contact mechanism would play any role. It would likely be much easier and much more effective to simply contact the programmer directly.

The effect of stars remains statistically significant, both inside and outside of the giant component. Modifications continue to be significant both inside and outside of the giant component. Additions, on the other hand, become insignificant outside of the giant component, but reamin significant inside of the giant component.

4.2 Projects with More than One Contributor

We repeat the analysis for projects with more than one contributor. Here, we again control for any potential endogeneity by restricting ourselves to projects with relatively few downloads. Columns 4 and 8 in Table 2 show that the main results discussed above continue to hold; thus, our results are robust to excluding projects with just a single contributor as well as restricting the analysis to projects with relatively few downloads.

Table 2: Results with Fixed Effects

DV: ldownloads	Outside Giant (1)	Outside Giant (2)	Outside Giant (3)	Outside Giant (4)	Inside Giant (1)	Inside Giant (2)	Inside Giant (3)	Inside Giant (4)
Constant	0.097 (2.39)	0.12 (3.72)	2.17 (69.43)	2.21 (35.68)	4.16 (10.47)	2.59 (12.21)	5.14 (32.16)	4.97 (24.62)
Lcpp	0.45 (41.43)	0.33 (25.60)	0.051 (4.20)	0.091 (3.13)	0.64 (53.31)	0.26 (20.18)	0.13 (10.08)	0.20 (11.18)
Ldegree	0.083 (7.54)	0.23 (34.97)	0.056 (12.90)	0.048 (6.47)	0.19 (9.38)	0.19 (18.78)	0.077 (10.02)	0.078 (8.50)
Lcloseness					1.70 (11.33)	0.47 (5.78)	0.92 (15.16)	0.74 (9.82)
Stars5	-0.14 (-5.50)	0.021 (1.92)	0.012 (1.81)	0.032 (2.63)	-0.17 (-6.82)	0.025 (2.89)	0.019 (3.08)	0.020 (2.61)
Stage	0.51 (107.3)	0.37 (43.45)	0.10 (12.74)	0.16 (10.48)	0.53 (74.81)	0.26 (26.44)	0.10 (10.79)	0.083 (6.92)
lyears_since	2.16 (115.0)	2.44 (360.14)	1.63 (291.8)	1.57 (136.2)	2.18 (72.15)	2.20 (230.52)	1.58 (184.16)	1.53 (134.76)
1add_total	0.026 (3.44)	0.022 (5.16)	0.0026 (0.65)	0.0071 (0.94)	0.036 (4.42)	0.022 (5.12)	0.014 (3.10)	0.017 (3.05)
lmod_total	0.19 (27.73)	0.040 (8.31)	0.021 (4.75)	0.029 (3.48)	0.17 (23.22)	0.033 (6.87)	0.019 (3.78)	0.016 (2.52)
Fixed Effects	NO	YES	YES	YES	NO	YES	YES	YES
Few downloads	NO	NO	YES	YES	NO	NO	YES	YES
cpp>1	NO	NO	NO	YES	NO	NO	NO	YES
# of Observations	107,534	107,534	60,126	15,690	53,608	53,608	30,678	18,019
Adjusted R-squared	0.24				0.38			

t statistics in parentheses

5. Analysis Using Differenced Data

In order to perform additional robustness checks, in this section we look at "difference" data from 2006 to 2009. The disadvantage of this approach is that we lose a large part of the

data, namely data from 2007 and 2008.[14] While the fixed effects analysis we conducted in sections 3 and 4 is preferable for this reason, the analysis in this section using difference data is an additional robustness check.

There are some projects that were not in the giant component in 2006 but were in the giant component in 2009. For these projects, we also include a dummy variable that takes on the value for projects that were in the giant component in 2009, but not 2006.[15]

Descriptive statistics for differenced data are shown in Table 6 in the appendix. The mean and median download changes for projects in the giant component (mean = 66,819 and median = 930) is much greater for projects in the giant component than for projects outside of the giant component (mean = 20,734 and median = 373). Correlations between changes in degree, closeness, Stars, and cpp (denoted Δdegree, Δcloseness, and ΔStar5) are all relatively low, as shown in Table 7 of the appendix. The highest correlation is between Δcpp and Δdegree, but that correlation is only 0.53. No other correlation exceeds a magnitude of 0.34.

5.1 Empirical Analysis using differenced data:

Here we run a difference on difference regression, where Δ means the difference between 2006 and 2009 and "ln" is the natural logarithm. Thus, we use the following estimating equation:

$$(6) \quad \ln(1+\Delta Downloads) = \beta_0 + \beta_1 \ln(1+\Delta cpp) + \beta_2 \ln(4+\Delta Degree) +$$

[14] Here, we also excluded observations for Δdegree, Δcloseness, and ΔCpp that are (approximately) in the lowest 5% of these distributions. Specifically, we exclude 961 observations of Δdegree that are less than or equal to -4, an additional 394 observations of ΔCpp that are less than or equal to -1, and an additional 557 observations of Δcloseness that are less than -0.0037. [14] After excluding these 1,912 projects, we are left with 40,884 observations for the analysis. The results are qualitatively unchanged if we include all observations.

[15] Our results are qualitatively unchanged if we eliminate the observations that were not in the giant component in 2006, but were in the giant component in 2009.

$$\beta_3 \ln(.0037+\Delta Close) + \beta_4(\Delta Star5) + \beta_5(\Delta Stage) + \beta_6 \ln(years\_since) +$$

$$\beta_7 \ln(1+ \Delta add) + \beta_8 \ln(1+\Delta mod) + \varepsilon,$$

Δstage is a dummy variable that takes on the value one if the projects moved from one stage to another during this period. ΔStar5 is a similarly defined dummy variable. The variable Δadd is the additions over the 2006-2009 period, less the additions that took place in 2006. The variable Δadd is similarly defined. We estimate (6) for the following cases:

Projects outside of the giant component in 2009

Column 1 – all observations,

Column 2 – projects with relatively few changes in downloads (in order to address issues of endogeneity of degree)[16]

Column 3 – projects with relatively few changes in downloads + cpp>1

Projects in the giant component in January 2009

Column 4 – all observations,

Column 5 – projects with relatively few changes in downloads

Column 6 – projects with relatively few changes in downloads + cpp>1

5.2 Projects Outside of the Giant Component

Columns 1, 2 and 3 in Table 3 show that changes in degree centrality are positively associated with changes in downloads for projects outside the giant component. The analysis using projects outside of the giant component also shows that modifications are positively associated with downloads, while additions are not. Further, for projects outside of the giant component, changes in the number of Stars are not significantly associated with downloads.

[16] We again exclude the upper 25% of the distribution of the change in downloads from 2006-2009.

These results are robust to restricting the analysis to relatively few downloads and projects with more than one contributor.

5.3 Projects inside the Giant Component

The analysis using projects inside the giant component (column 4) shows that both changes in degree and changes in closeness are positively associated with changes in downloads. As in the analysis using fixed effects, the estimated coefficient is smaller on Δdegree (and Δcpp) when we restrict the analysis to projects with relatively few downloads, suggesting that degree (and cpp) are endogenous. Inside of the giant component, degree is no longer statistically significant. In the case of closeness, we find that the estimated coefficient is virtually unchanged and statistically significant when we restrict the analysis to projects with few downloads, suggesting that again, as in the fixed effects case, closeness is not endogenous.

The analysis also shows that additions are not statistically associated with downloads, neither outside nor within the giant component. Modifications on the other hand, are positively associated with downloads across all specifications, both inside and outside of the giant component.

5.4 Summary

In summary, using the "long" difference analysis (looking at changes from 2006 from 2009,) we find that the following key results are qualitatively unchanged from the fixed effects analysis.

- Closeness is significantly associated with success, suggesting that there are both direct and indirect spillovers.
- Modifications are significantly associated with success, suggesting that incremental innovation is quite important.

Table 3: Results using differenced data

DV: lΔdownloads	Outside Giant Component (1)	Outside Giant Component (2)	Outside Giant Component (3)	In Giant Component (4)	In Giant Component (5)	In Giant Component (6)
Constant	6.27 (33.32)	6.14 (39.05)	5.73 (19.11)	4.97 (15.97)	6.80 (24.72)	6.59 (17.64)
lΔCpp	0.57 (8.12)	0.20 (2.66)	0.37 (3.92)	0.65 (11.89)	0.18 (2.75)	0.18 2.58)
lΔdegree	0.32 (5.59)	0.30 (6.17)	0.25 (3.17)	0.14 (2.95)	0.033 (0.75)	0.034 (0.65)
lΔcloseness				0.10 (3.26)	0.12 (4.60)	0.11 (3.23)
ΔStars5	-0.049 (-0.79)	-0.012 (-0.23)	-0.041 (-0.43)	0.071 (1.32)	0.10 (2.14)	0.049 (0.45)
Δstage	0.45 (7.49)	0.24 (4.28)	0.12 (1.06)	0.23 (2.84)	0.16 (1.91)	0.16 (2.53)
lyears_since	-0.62 (-9.10)	-0.75 (-13.46)	-0.54 (-4.67)	0.58 (5.50)	-0.22 (-2.41)	-0.084 (-0.67)
Moved into Giant				-0.42 (-4.93)	-0.33 (-4.55)	-0.30 (-3.03)
Ladd_total	0.012 (0.74)	-0.020 (-1.26)	-0.031 (-0.43)	-0.00010 (-0.05)	-0.0084 (-0.47)	0.022 (1.03)
Lmod_total	0.30 (19.74)	0.12 (8.27)	0.12 (4.84)	0.35 (22.23)	0.17 (10.97)	0.13 (6.66)
Few downloads	NO	YES	YES	NO	YES	YES
Cpp>1	NO	NO	YES	NO	NO	YES
# of Observations	27,410	20,554	5,357	13,474	10,106	5,784
Adjusted R-squared	0.10	0.03	0.04	0.25	0.07	0.07

6. Conclusion:

Employing a simple model of knowledge spillovers, we find empirical evidence among open source software projects that is consistent with both direct and indirect spillovers. This result is robust to running the analysis using a fixed effect model or a difference model, i.e., looking at differences over the 2006 to 2009 period. There is also evidence that the addition of a "star" programmer on the project is associated with greater success, even after controlling for the induced change in the network structure (more directs connections with other projects) from the addition of a star. Finally, we find that modifications are significantly associated with project success and this result is extremely robust. This suggests that "incremental" innovations are critical to project success.

References

Ahuja, G. 2000. Collaboration Networks, Structural Holes, and Innovation: A Longitudinal Study. Adm. Sci. Q. 45(3) 425–455.

Angrist, J., and J. Pischke, 2009, "Mostly Harmless Econometrics," Princeton University Press, Princeton, New Jersey.

Ballester, C., A. Calvó-Armengol, Y. Zenou. 2006. Who's who in networks. Wanted: the key player. *Econometrica* **74**(5) 1403–1417.

Bonaccorsi, A., C. Rossi, S. Giannangeli. 2006. Adaptive entry strategies under dominant standards: Hybrid business models in the Open Source software industry. Manag. Sci. 52(7) 1085–1098.

Calvo-Armengol, A., M. O. Jackson. 2004. The effects of social networks on employment and inequality. *Am. Econ. Rev.* **94**(3) 426–454.

Calvó-Armengol, A., E. Patacchini, Y. Zenou. 2009. Peer effects and social networks in education. Rev. Econ. Stud. 76(4) 1239–1267.

Fershtman, C., & Gandal, N. 2011. Direct and indirect knowledge spillovers: the "social network" of open-source projects. The RAND Journal of Economics, 42(1): 70–91.

Freeman, L. 1979. Centrality in social networks: Conceptual clarification. Soc. Netw. 1(3) 215–239.

Goyal, S., M. J. Van Der Leij, J. L. Moraga-González. 2006. Economics: An emerging small world. J. Polit. Econ. 114(2) 403–412.

Grewal, R., G. L. Lilien, G. Mallapragada. 2006. Location, location, location: How network embeddedness affects project success in open source systems. Manag. Sci. 52(7) 1043–1056.

Hippel, E. von, G. von Krogh. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. Organ. Sci. 14(2) 209–223.

Jackson, M. O., L. Yariv. 2007. Diffusion of behavior and equilibrium properties in network games. *Am. Econ. Rev.* **97**(2) 92–98.

Karlan, D., M. Mobius, T. Rosenblat, A. Szeidl. 2009. Trust and social collateral. *Q. J. Econ.* **124**(3) 1307–1361.

Laurent, A. M. 2004. Understanding open source and free software licensing. O'Reilly Media, Inc.

Lerner, J., J. Tirole. 2002. Some simple economics of open source. J. Ind. Econ. 50(2) 197–234.

Lipnack, J., J. Stamps. 1997. Virtual teams: Reaching across space, time, and organizations with technology. John Wiley & Sons Inc.

Wasserman, S. 1994. Social network analysis: Methods and applications. Cambridge University Press.

Appendix:

Table 4: Descriptive Statistics for Analysis in Sections 3 and 4

Outside of Giant Component

Variable	Obs	Mean	Std. Dev.	Min	Max
downloads	107534	18838.9	829788	1	1.85e+08
years_since	107534	5.008747	1.922073	.9664613	10.15195
degree	107534	1.267357	2.039043	0	33
cpp	107534	1.627606	1.566627	1	53
stars5	107534	.0837224	.2769723	0	1
stage	107534	3.758541	1.160229	1	6
add_total	107534	72.52826	799.6828	0	91752
mod_total	107534	111.8213	2294.844	0	469000

Inside of Giant Component

Variable	Obs	Mean	Std. Dev.	Min	Max
downloads	53608	96998.17	3269947	1	4.74e+08
years_since	53608	5.711708	1.93483	.9691992	10.16016
degree	53608	6.847392	8.534879	1	264
closeness	53608	.0331818	.0049721	.0131664	.0518591
cpp	53608	4.071687	6.946684	1	267
stars5	53608	.4974258	.499998	0	1
stage	53608	3.944318	1.139927	1	6
add_total	53608	271.1987	2164.149	0	103368
mod_total	53608	505.7636	3324.277	0	209882

Table 5: Correlation among Centrality Variables (Giant Component, N=53,608)

	cpp	degree	closeness	Star
cpp	1.00			
Degree	0.62	1.00		
Closeness	0.27	0.44	1.00	
Star	0.14	0.49	0.30	1.00

Table 6: Descriptive Statistics for "Difference" analysis in section 5

	Variable	*Observations*	*Mean*	*Std. Dev.*	*Min*	*Max*
Projects outside the giant component	Δdownload	27410	20733.59	1109424	0	1.71e+08
	ΔDegree	27410	-.0322875	1.134059	-4	19
	ΔCpp	27410	.0694637	.5928583	-1	20
	ΔStage	27410	0.0444728	.2061469	0	1
	ΔStars5	27410	-.0048887	.2240058	-1	1
	years_since	27410	6.57	1.55	3.97	10.15

	Variable	*Observations*	*Mean*	*Std. .Dev.*	*Min*	*Max*
Projects in the giant component	Δdownload	13,474	66,819	1,880,295	0	1.98e+08
	ΔCloseness	13,474	.000059	.012	-.00369	.0454
	ΔDegree	13,474	0.92	3.85	-4	103
	ΔCpp	13,474	.64	3.15	-1	104
	ΔStage	13,474	0.058	0.23	0	1
	ΔStars5	13,474	.041	.38	-1	1
	years_since	13,474	7.14	1.64	3.97	10.16

Table 7: Correlation among Centrality Variables Difference Analysis (Giant Component: N=13,474)

	ΔCpp	Δdegree	Δcloseness	Star
ΔCpp	1.00			
ΔDegree	0.53	1.00		
ΔCloseness	0.06	0.18	1.00	
Star	0.09	0.34	0.21	1.00