Corporatizing Open Source Software Innovation in the Plone Community

George Kuk
Nottingham University Business School
g.kuk@nottingham.ac.uk

Guido Stevens
Cosent
guido.stevens@cosent.net

ABSTRACT

Increasingly open source (OS) software development is organized in a way similar to how a corporation would organize development. This paper examines this corporatizing effect by studying the relationship between peer-oriented social structures and goal-oriented technical structures in the Plone community. Social structures are said to exhibit assortative mixing, a like attract like characteristic whereas technical structures exhibits an opposite effect of disassortative mixing. Our first finding suggests that the patterns of collaborative contributions and interdependences among software modules exhibit the characteristic of disassortative mixing. Specifically, Plone developers were more likely to contribute to modules that already have a high concentration of contributions, which in turn lead to an increase in module reuse over time. This finding contributes to the debate of whether social systems are strictly assortative, and technological systems strictly disassortative (Newman, 2002). Our second contribution concerns the impact of corporatizing OSS projects, suggesting that corporatizing OS development had the effect of weakening the social organizing among developers, and shifted the patterns of contributions to adhere with the technical requirements.

Keywords (Required)

Assortativitive mixing, disassortative mixing, corporatized innovation, open source

INTRODUCTION

Central to the scholarships of organization science is the question of how to organize individuals to use and generate resources. Recent research has taken a view that individuals are organized around the products they produce, and the work structures are organized and channels of communication are designed around technical considerations to facilitate social interactions (e.g. Hoetker, 2006; Sosa, Eppinger & Rowles 2004; Sosa, 2008). Yet the open source software (OSS) development has challenged this view. It takes a different perspective of how individuals self-organize and govern their work practices. The parallel of this is perhaps the mostly used contrast between the bazaar and the cathedral styles of software development (Raymond 1999). The former concerns the altruistic and strategic behaviors of hobbyist developers, collaboratively contributing to software development. It portrays a democratized process, involving grassroots participation driven by core developers (von Hippel and von Krogh 2003) whereas in the latter, technical requirements and corporate interests take precedence over individual developers' preferences and group consensus. As commercial firms increase sponsorships of open source projects, two immediate questions arise: How does corporatizing a democratized process work? What are impacts on corporatization on technical and social structures of OSS development?

This paper seeks to understand this intricate relationship by examining structure and agency in OSS development. The structure is referred to as the interdependencies among software modules and that the agency as the actual performances carried out in time and place by particular developers. We compare and contrast the observed structures of module dependency that adhere to technical requirements with their counterparts that are based on how developers organize their contributions. Most research has studied technical and social structures as separate entities, and often in samples rather than considering the entire studied population. This paper dovetails from this by examining the code dependency among modules in the Plone community between 1997 and 2008.

Theoretical Background

How developers organize among themselves and contribute to OSS development has been studied at both macro and micro-levels. In a survey of OSS projects registered on the Sourceforge, Madley et al. (2004) found that only a small percentage of OSS projects received contributions from the majority of developers. This suggests that OSS developers tend to gravitate towards a few, most popular projects. This underlines the rich get richer effect in networks. Kuk (2006) in his study of the KDE developer mailing list found that concentration served to alleviate the problems of coordination overheads especially when everyone was trying to contribute. But the concentration was beneficial up to a limit and beyond that, knowledge sharing begun to suffer. Both studies draw upon the notion of preferential attachment but subtle differences exist between the two.

The macro study by Madley et al. concerns the so-called disassortative mixing, which in network terms, depicts the tendency of a node of a low degree of connection to attempt to connect to a node of a higher degree, and/or vice versa. For instance, when new comers self-select either to work in projects with a high concentration of developers or to solicit collaborative working with established developers. Another possible scenario involves senior members actively seeking new recruits. The micro study by Kuk concerns a direct opposite, commonly known as assortative mixing, in that nodes of similar degrees tend to attract each other. This is particularly true as high status, resourceful individuals tended to collaborate more often among themselves in assortative co-authorship networks (Newman 2002).

Von Krogh et al (2003) develop a script theory to reflect the joining mechanism of new comers in the Freenet community. Their qualitative case study suggests new comers have to be first accepted by the existing members before they can move up the status hierarchy of an established community. The script theory adheres to the principle of assortative mixing. It underscores individuals are organizing according to the social etiquettes of a status hierarchy (Brown & Duguid, 2001). A core goal is to facilitate social interactions such that new comers will only attempt to connect with existing members of a similar status.

Yet a recent longitudinal study of an online social network has suggested that the network was only predisposed to assortative mixing at the initial stage. Over time it was shifted to disassortative mixing as the increase of connections of senior members with others including new comers served to signal and reinforce status. Unlike network ties in professional collaborations in academic and business networks, disassortative mixing was less costly to maintain (Hu & Wang, 2009).

We suggest that the mixed findings of how individuals are organizing, whether assortatively or disassortatively, are subject to three important considerations. First, Lee and his associates (Lee, Kim & Jeong, 2006) have shown different sampling procedures can result in different mixing patterns, considering that the selection criteria are largely arbitrary. Second, the studied samples in the above literature often presented a snapshot view rather than its entirety relating to the patterns of contributions, and growth of the studied community. Third, how individual developers organize may be highly strategic. They may pursue assortative initially and disassortative mixing at a later stage. The initial assortative mixing follows the theory of collective action that a small group of highly motivated and resourceful individuals tends to work closely together to bear the start-up cost in collectively producing public goods such as OSS (Marwell & Oliver 1993). This strategic characteristic is important to understand the organizing dynamics that the individual developers exhibit over time; and that initial structures facilitates the creation of additional resources by co-opting new comers at a later stage.

In view of the above, we propose that the organizing dynamic at the initial growth stage of an OSS community is likely to be assortative. Kuk (2006) argues that assortative mixing helps to the building of reciprocal relationships and trust among a smaller group of co-developers, but also optimizes better use of heterogeneous resources that each member brings without duplicating effort. As the project expands, it attracts more new comers through disassortative mixing and/or involves the senior members soliciting support from the new comers which is fairly common in the OSS communities[1]. This has led to the following hypothesis:

> *Hypothesis 1: The organizing dynamic among software developers is more likely to be assortative at the initial stage of the OSS project, and followed by the pattern of disassortative mixing.*

If new comers are organizing to increase their chance of interacting with senior members of the community, an obvious strategy is to try to contribute to an existing, popular module, or reuse blocks of code by linking with an existing module in their pursuit of newer projects. Code reuse not only costs less in terms of the software development and maintenance but also facilitates knowledge integration (Haefliger, von Krogh, and Spaeth, 2008). The ways individuals are organizing around popular modules, and reusing codes, are likely to promote disassortativity among modules and codes. This pattern reiterates

[1] For example, in the Plone community, new members are often recruited to work on various projects through boot camps, sprints and regular events.

the importance of code reuse in OSS development. It is likely that newer projects are more likely to reuse rather than develop code from scratch. The disassortative mixing in code reuse is demonstrated empirically in a study of code dependency in the open source Linux distribution (Maillart et al., 2008). The above consideration has led to the following two hypotheses.

> *Hypothesis 2a. With the ways individual developers strategically organize their contributions, the module interdependencies linked directly to developers' contributions are likely to be assortative initially and disassortative over time.*
>
> *Hypothesis 2b. With greater code reuse between older and newer modules, the module interdependencies adhered to technical requirements are likely to be assortative when there was few established modules, but disassortative when more new modules are introduced over time.*

If OSS development is a social process, driven by grass root participation and lead developers, then we expect network ties formed among developers will strengthen code reuse, and bring disparate modules closer together. This follows the argument that structure and agency are recursively constitutive. Orlikowski and Barley (2001) suggest that individuals do not passively consume technologies, but actively use them to either support or modify the social structures around work. In proprietary software development, code reuse and the interdependencies among modules can be specified a priori in adherence with the technical requirements, allowing a smaller group of developers to manage the process. Although the process of OSS development is more fluid and malleable through group consensus (Scacchi, 2004), the development process often is managed and concentrated on a small group of core developers (Mockus, 2002). With the development process managing by a small group of developers, it is likely that both social structures in closed and open software development will exert a similar effect on how modules are linked and developed initially. This has led to the following hypothesis.

> *Hypothesis 3. The social structures formed as a result of network ties among a small group of developers will determine how software modules are related to each other. That is, the technical structures are causally determined by the social structures.*

Many OSS projects have huge commercial values. They have attracted increased corporate sponsorships. Although firms can always donate codes and initiate a new project, the majority has chosen to join an existing community (Dahlander, 2007). For firms involving in OSS, their involvement in defining structures, and organizing and managing resources are inevitable (Dahlander & Magnusson, 2005). This has led to increased corporatization, addressing the integration challenges of buy-in innovation that firms encounter in open sourcing the development of software and services. The corporatization effect can undermine the social influence on the organizing dynamic in OSS development, and change it to a managed process similar to the process of proprietary software development. The reverse is also true, as many firms across many industries have perceived and acted upon the necessity of designing more open approaches to value creation and value capture. Open sourcing can open up the design process, imposing fewer restrictions on what developers can do, and with whom they can collaborate. But as the size of the community, and the complexity and the scope of the projects increase, technological concerns will supersede the social structures. This has led to our final hypothesis.

> *Hypothesis 4. Corporatizing OSS projects will align the organizing dynamics among developers to follow the technical requirements. That is, technical structures will ultimately determine the social structures to coincide with an increase in the size, and scope and complexity of the projects.*

RESEARCH APPROACH

We studied a number of open source website development platforms written in the Python programming language. Python is an open-source, interpreted programming language which provides both rich language syntax and a deep library of reusable code building blocks. The biggest Python web framework is the Plone Content Management System (CMS). Plone is community owned; trademarks and key copyrights are held by the Plone Foundation. Although it is community owned, the organizing and managerial aspects of Plone follow the corporatized and market regime than individual users and hobbyists' preferences (Aspeli, 2004). This provides a rich research setting to examine the effect of corporatizing OSS development on technical and social structures.

Code Ecology

The Plone system comprises about one million lines of source code including the Zope application server, on which Plone is built. Understanding and managing a system of such magnitude requires a divide-and-conquer approach. The organizing pattern utilizes a generic framework calling specialized plugin components. Control resides with the framework, which determines execution flow, not with the plugin.

A Plone website is created by crafting an interrelated set of custom plugins that modulate visual appearance, define information schemata, and control policy behaviors to adapt the generic system to client-specific requirements. Integrators doing Plone customizations can draw upon an extensive library of plugin components providing specialized functionality which is not included in the core Plone framework "out-of-the-box". Such generic components are created by other integrators, who encountered similar requirements and published their solutions to be re-used.

Underlying the custom and generic plugins, is the Plone core system, consisting of central infrastructure components augmented by dozens of specialized aspect providers. Plone itself plugs into the Zope framework, which also can be deconstructed as a set of core infrastructures augmented by specialized components. The whole of this component architecture is configured and integrated to act as a single, integrated system.

Corporatizing Work Practices

The fall 2009 Plone conference saw an attendance of circa 400 Plone developers. A typical Plone developer is employed as such by an IT services company providing integration and customization services to customers. As of January 2010, the central directory of Plone integration providers listed 328 "Plone providers" in 60 countries worldwide.

Plone developers are generally IT professionals, often formally trained in computer science, who are paid to work with Plone (for customers), and for whom working on Plone (for the community) is a normal application of their skills. Several well-established Plone integration providers subscribe to an informal policy of donating 10% of employee time to the Plone community in the form of open source software contributions.

The Plone CMS provides a bundle of features that is on par with commercially developed competitors. Plone can be downloaded for free, and provides prospective client organizations with a compelling value proposition: it offers both a feature-rich CMS environment "out of the box" as well as excellent customization options. The availability of a mature market of Plone integration providers, in combination with the open source aspect, is an important consideration for many organizations that want to minimize the risk of lock-in to a specific technology provider.

Data Collection

The data collected covers the full revision history of the Zope/Plone system from July 1996 onwards, supplemented with data from other Python-based web development platforms. Software codes are stored, managed and published using version control systems that provide central coordination services to facilitate collaboration between developers. All systems included in this study, used Subversion servers to manage software development.

We sent requests to the relevant mailing lists. The following Subversion repositories were then made available in the form of a full svn dump mirror: zope (Zope application server), plone (Plone core), archetypes (Plone core content types), and collective (Plone add-on plugins). These repositories were loaded into a local Subversion server. Using git-svn, the Subversion repositories were imported into Git, another version control system with features that support the type of analysis performed in this study. The data obtained in this way were supplemented by performing git-svn crawls against the public Subversion servers of Django, Repoze and Turbogears. Some Plone components hosted outside the Plone Subversion server were directly imported from the codespeak.net Subversion server.

The data collected covers full code histories of multiple Python web development communities with diverse degrees of interrelationship, both in terms of code re-use and community overlap. However, the sizes of the respective code bases are very unevenly distributed. After conversion to Git, core Zope/Plone components (that are present in all Plone 4.x installations) account for 42.1% of code base disk space, while Plone addons (optional Plone components) claim 53.9% of disk space. Repoze uses 2.6% of disk space, leaving only 1.4% for Turbogears and Django combined. Hence, even though the data collection was designed to comprehensively cover a variety of sources, the collected data overwhelmingly (98.6% of disk space) reflects the Zope/Plone family of software codes.

Measures

Author: an author is the account id used to log in to a Subversion server and contribute software changes. It is likely that a real person will create and use the same author id on different Subversion servers.

Module: a separate unit of Python code, contained in a file.

Package: a collection of Python modules, organized into a directory tree. From a programming perspective, packages also behave like modules.

Dependency: an import linkage from one module to another. Modules import functionality from other modules. Only by importing is the imported module visible to the importing module, but not vice versa. Such highly constrained code pathways allow for the construction of complex yet comprehensible systems.

SLOC: a source line of code. Version control systems manage and record changes at the source line of code level. The SLOC count can be used as a proxy for the amount of effort invested in, or functionality available from, a body of source code.

Commit: a bundle of changes to source code lines, registered with a version control system by an author. Commits record changes at the SLOC level and bundle changes at the package level.

Revision: the state of a package that results from a commit.

Snapshot: the revision state of a package at a predefined moment in time.

Revision history: the chronology of revisions for a specific package. On a specific point in time, multiple revisions may be valid in the form of branches. In this study, the revision whose commit time stamp most closely predates a snapshot time stamp is chosen as "the" revision for that snapshot.

Network Extraction

The analysis was restricted to Python source code files. This prevents SLOC count distortions from large test data files; machine generated quality assurance artifacts do not reflect programmer agency in a way that is comparable to hand-crafted software codes. Also, dependency analysis is only possible for software codes, I.e. Python files; configuration and documentation text files describe the accompanying Python logic and do not add code dependencies themselves. For each package, the full revision history was rewound commit by commit. At each month boundary, a snapshot was created to reflect the revision state for that package on the start of that month. For each monthly snapshot, authorship was analyzed at the SLOC level. That is, each line of source code was assigned to the author that had last edited that line at the snapshot boundary. This resulted in SLOC counts per author per module per snapshot for each package. Additionally, for each monthly snapshot, dependencies between modules were extracted using regular expression matching. This resulted in dependency listings of all imported modules per importing module per snapshot for each package. Some of the growth indicators of the Zope/Plone CMS community are given in Table 1.

Table 1. Some Growth Indicators of the Plone CMS Community

Year[a]	SLOC	Modules	Packages	Authors
1998	18861	76	2	7
1999	49126	187	2	8
2000	108728	353	3	14
2001	133343	447	3	23
2002	177077	797	4	39
2003	335083	2099	7	70
2004	484386	2966	21	135
2005	714725	3950	33	176
2006	1247768	7581	203	257
2007	1487448	9598	415	363
2008	1517348	11260	624	411

Note: [a]We only included years with full 12 months of data

Network Structures, Measures and Analyses

Network structures. We used the information about affiliations among modules and authors to construct our two network structures: first, the technical network, representing the interdependencies among modules based on the sharing of the same block of code; and second, the social-technical network, representing the interdependencies among modules based on the sharing of the same author. An affiliation network, commonly known as a two-mode network (Wasserman and Faust, 1994), defines the relationships between actors and events, for example, authors represent the actors and modules represent the events in the social-technical network.

Network measures. We computed the clustering coefficients of all individual software modules over the period 1997 to 2008. Clustering coefficients are typically used for gauging the patterns of assortative and disassortative mixing by plotting node degree against clustering coefficient. Assortative mixing will review a positive Pearson correlation whereas disassortative will give a negative Pearson correlation.

Network Analysis. In order to test Hypothesis 1, the social-technical network was transformed to a single-mode network that contained only co-authorships information. To test hypotheses 2a and 2b, the two-mode networks were transformed to single-mode networks that contained module dependencies only. To test hypotheses 3 and 4, the single-mode technical and social-technical networks were used over the period 1997 to 2008. We deployed the method of cross-lagged panel correlation analysis (Kenny 1975, 1979), which involved testing of the cross-lagged correlations of two variables over at least two consecutive time points. The variables comprised the clustering coefficients of software modules based on technical vs. social-technical networks. The uses of clustering coefficients are typically used for gauging the assortative and disassortative trends by plotting node degree against clustering coefficient.

Figure 1 illustrates the cross-lagged design, the four measures resulted in two cross-lagged correlations (rStTt+1,, rTtSt+1), two autocorrelations, and two synchronous correlations. The two cross-lagged correlations can be expressed as a cross-lagged differential (rStTt+1 – rTtSt+1). When assumptions of synchronicity, stationarity, and equal stability are met, a cross-lagged analysis provides a test for spuriousness, that is, of whether or not the relationship between social-technical and technical networks from the effects of some third, unspecified, variable rather than the causal effects of either.

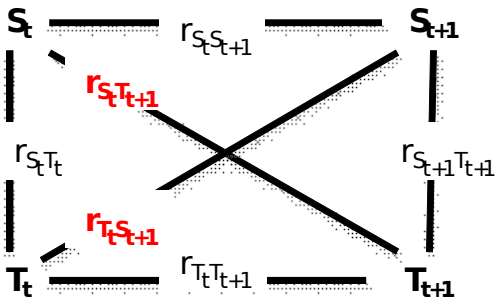

Figure 1. A cross-lagged panel correlation design for social-technical (S) and technical (T) network for two consecutive time points.

RESULTS

Table 2 reports the Pearson correlation coefficients between node degrees and clustering coefficients in the co-authorship networks over the period 1997 to 2008. As the table shows, between 1997 and 2004, there were three significant patterns of assortative mixing (year: 1999, 2000, 2004) and two significant patterns of disassortative mixing (year: 1997 and 2003), and from 2004 onwards, three periods exhibited significant disassortative mixing (year: 2005, 2006 and 2007). The results support Hypothesis 1, in that the majority of the initial organizing dynamics were more assortative, and followed by disassortative mixing.

Table 2. Co-authorship Network: Correlations between Node Degree and Clustering Coefficients

Year	N	Pearson correlation
1997	7	-0.88***
1998	9	-0.18
1999	14	0.68***
2000	23	0.40*
2001	38	-0.15
2002	79	-0.19
2003	127	-0.73***
2004	179	0.87
2005	266	-0.30***
2006	374	-0.11*
2007	488	-0.09*
2008	522	0.01

By contrast, the majority of the module dependencies within the technical and social-technical networks were disassortative (as shown in Table 3). In the social-technical networks, there were ten significant patterns of disassortative mixing (between 1999 and 2008), and only one significant assortative mixing in year 1998. For the technical networks, there were two significant assortative mixing in years 1997 and 2000, showing that popular modules tended to be reused more often among themselves. Over time, the new modules (introduced between 2002 and 2008) tended to continue reusing the existing, and popular modules. The results support hypotheses 2a and 2b, that initially when there were only a few, established modules, developers tended to gravitate around them, this gave rise to assortative mixing. Over time, these popular modules tended to be reused more often especially when new modules were introduced over time, this accounted for the observed disassortative mixing.

Table 3. Correlations between Node Degrees and Clustering Coefficients

Year	N	Social-technical Networks	Technical networks
1997	72	-0.03	0.25*
1998	159	0.48***	0.11
1999	331	-0.23***	0.09
2000	385	-0.49***	0.24***
2001	762	-0.72***	0.01
2002	1939	-0.34***	-0.22***
2003	2844	-0.14***	0.07***
2004	4315	-0.34***	-0.29***
2005	7376	-0.52***	-0.49***
2006	9589	-0.47***	-0.49***
2007	13273	-0.61***	-0.48***
2008	13273	-0.46***	-0.08***

Table 4. Pearson-Filon Comparisons of Cross-Lagged Correlations Between Social and Technical Networks

Year	N	$r_{S_tT_{t+1}}$		$r_{T_tS_{t+1}}$		z	*p*
1997 - 1998	66	-0.14		0.07		-1.09	
1998 - 1999	150	0.27	***	0.21		0.97	
1999 - 2000	309	0.16	**	0.12	*	0.74	
2000 - 2001	374	0.15	**	0.05		2.17	**
2001 - 2002	671	0.16	***	0.06		2.52	**
2002 - 2003	995	-0.05		0.06		-2.61	**
2003 - 2004	2395	0.20	***	0.08	***	4.20	***
2004 - 2005	3187	0.20	***	0.23	***	-2.07	*
2005 - 2006	6200	0.28	***	0.33	***	-5.71	***
2006 - 2007	7940	0.29	***	0.30	***	-2.55	***
2007 - 2008	8986	0.18	***	0.22	***	-3.48	***

Table 4 reports the Pearson-Filon tests of the cross-lagged correlations between the clustering coefficients of the social-technical and the technical networks. Prior to 2003, our dataset recorded mainly Zope codes. From the beginning, Zope was privately owned, and open sourced in 1998. Significant causal relationships were found starting with the period 2000 – 2001 onwards. Specifically, the cross-lagged correlations in the periods 2000-2001, and 2001-2003, were significant and positive, suggesting that the initial open sourcing affects the ways developers organizing among themselves and significantly determined the technical structures among module relationships. This supports hypothesis 3. Table 4 also shows from the period 2004-2005 onwards, the corporatized innovation in response to commercial requirements had reversed the causal impact of social on technical structures. The results supported Hypothesis 4, indicating technical factors affects how

developers organizing to coincide with an increase in the size, scope and complexity of the Plone projects. The above findings were found to be robust as the testing of the assumptions of synchronicity, stationary, and stability were all satisfied.

DISCUSSION

Our findings reflect the histories and the organizing dynamics within the Plone community. The progression is one of starting out small, with a compact developer team crafting a set of codes to solve a specific challenge. At a certain point, the value contained in the open source codes is significant enough to attract a developer community to reuse the codes as a spring board for creating new functionalities. The values added to the software compound over time, and as more and more programming talents gravitate towards the community, the size and complexity of the code base increases and the size of, and differentiation in the social community increase.

The initial assortative mixing makes sense where few people closely collaborate on a limited code set. As time progresses and complexity increases, disassortativity becomes a way of managing technical complexity by structuring inter-code linkages in the way described above under "code ecology". The progression of time creates a growing gap in experience between new comers and the old guard. Growing technical complexity presents an ever steepening learning curve for new comers. The old guard has experienced the evolution of the software through time and occupies a much better knowledge position; also the old guard network will be much better connected socially. These trends combine to give rise to disassortative social structures, which allow for resource efficient coordination in the face of complex structures (Kuk, 2006).

As scale and complexity increase, both technical structures and social structures gravitate to disassortative core/periphery patterns of connection. Corporatized innovation in response to the commercial needs in an open source environment provides additional resources and incentives for the developers to contribute and collaborate. It creates a marketplace for the Plone integrators to perform knowledge arbitrage against their paying customers. A position of superior knowledge about the inner workings and customization flex points of the Plone system allows Plone integrators to create customized Plone-based CMS solutions that deliver high business values to client organizations, for whom it is more efficient (cheaper, faster) to buy a solution than to build up the required competencies in-house.

Even though integrators compete against each other within certain markets, collaboration provides competitive benefits. Sharing knowledge and software codes between Plone integrators improves their knowledge position and hence their capability to deliver value to paying customers. Any improvement in Plone itself, either as an improvement to the core, or as a plugin component, or in the form of better documentation or marketing efforts, improves the overall Plone value proposition and the competitive position of Plone vis-a-vis competing platforms, e.g. Java-based CMS systems.

The patterns of assortative and disassortative mixing explain some of the virtues of open source innovation. At the formation stage of the OSS community, assortativity gives the community resilience (Newman, 2002), and as the size of the community increases coupled with an increase in scope and complexity of the projects, disassortivity provides the needed mechanisms for the spreading of innovation through the reuse of code, and increase module dependency, which in turn expedites innovation incrementally.

REFERENCES

1. Brown, J.S., & Duguid, P. 2001. Knowledge and organization: a social-practice perspective, Organization Science, 12: 198-213.
2. Dahlander, L. 2007. Penguin in a new suit: a tale of how de novo entrants emerged to harness free and open source software communities. Industrial & Corporate Change, 16: 913-943.
3. Dahlander, L., & Magnusson, M. G. 2005. Relationships between open source software companies and communities: Observations from Nordic firms. Research Policy, 34: 481-493.
4. Emirbayer, M., & Mische, A. 1998. What is Agency? American Journal of Sociology, 103:962-1023.
5. Grand, S., Von Krogh, G., Leonard, D., & Swap, W. 2004. Resource Allocation Beyond Firm Boundaries: A Multi-Level Model for Open Source Innovation. Long Range Planning, 37(6), 591-610.
6. Haefliger, S., von Krogh, G., & Spaeth, S. 2008. Code reuse in open source software. Management Science, 54: 180-193.
7. Haruvy, E., Prasad, A., & Sethi, S. P. 2003. Harvesting altruism in open source software development. Journal of

Optimization Theory and Applications, 118: 381-416.

8. Hoetker, G. 2006. Do modular products lead to modular organizations? Strategic Management Journal, 27, 501-518.
9. Hu, H. B., & Wang, X. F. 2009. Disassortative mixing in online social networks. A letter Journal Exploring the Frontiers of Physics. 86:18003.
10. Kenny, D. A. 1997. Correlation and causality. New York: Wiley.
11. Kenny, D. A. Cross-lagged panel correlation: a test for spuriousness. Psychological Bulletin, 82: 887-903.
12. Kleinberg, J. 2008. The convergence of social and technological networks. Communications of the ACM, 51(11), 66-72.
13. Kuk, G., "Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List," Management Science (52:7), 2006, 1031-1042.
14. Lee, S. H., Kim, P. J., & Jeong, H. 2006. Statistical properties of sampled networks. Physical Review E, 73:016102.
15. Madey, G., V. Freeh, R. Tynan. 2004. Modeling the F/OSS community: A quantitative investigation. S. Koch, ed. Free/Open Source Software Development. Idea Publishing, Hersey, PA, 203–220.
16. Maillart, T., Sornette, D., Spaeth, S., & von Krogh, G. 2008. Empirical tests of Zipf's law mechanism in open source Linux distribution. Physical Review Letters, 101: 218701.
17. Newman, M. E. J. 2002. Assortative mixing in networks. Physical Review Letters, 89: 208701.
18. Orlikowski, W. J. 2000. Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations. Organization Science, 11: 404-428.
19. Orlikowski, W.J. & Barley, S.R. (2001). Technology and Institutions: What Can Research on Information Technology and Research on Organizations Learn from Each Other? MIS Quarterly, 25, 145-165.
20. Raymond, E. 1999. The cathedral and the bazaar. First Monday. Retreived April 14, 2004, http://www.firstmonday.org/issues/issue3_3/raymond/.
21. Scacchi, W. 2004. Free and open source development practices in the game community. IEEE Software, 21:56–66.
22. Sosa, M. A., Eppinger, S. D., & Rowles, C. M. 2004. The misalignment of product architecture and organizational structure in complex product development. Management Science, 50: 1674-1689.
23. Sosa. M. A. 2008. A structured approach to predicting and managing technical interactions in software development. Research Engineering Design, 19: 47-70.
24. Stewart, D. 2005. Social Status in an Open Source Software Community, American Sociological Review, 70: 823-842.
25. von Hippel, E., & von Krogh, G. 2003. Open source software and the "private-collective" innovative model. Organization Science, 12: 209–223.
26. von Krogh, G., S. Spaeth, & Lakhani, K. R. 2003. Community, joining, and specialization in open source software innovation: a case study. Research Policy, 32: 1217–1241.
27. Wasserman, S., & Faust, K. 1994. Social network analysis: methods and applications. Cambridge University Press.
28. Zeitlyn, D. (2003). Gift economies in the development of open source software: anthropological reflections. Research Policy, 32: 1287-1291.