2010

Eyeballs, Bugs, and Releases in Open Source Software

George Kuk
Nottingham University, g.kuk@nottingham.ac.uk

EYEBALLS, BUGS, AND RELEASES IN OPEN SOURCE SOFTWARE

Journal:	*18th European Conference on Information Systems*
Manuscript ID:	ECIS2010-0336
Submission Type:	Research Paper
Keyword:	OSS/FLOSS, User involvement/participation, Networks, Network organizations

EYEBALLS, BUGS, AND RELEASES IN OPEN SOURCE SOFTWARE

Kuk, G, Nottingham University Business School, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK, g.kuk@nottingham.ac.uk

Abstract

This study examines two widely cited principles of Linus' law, namely "given enough eyeballs, all bugs are shallow", and "release early and release often". The aim is to understand their relationships and their limits to debugging open source software (OSS) bugs. Anecdotally, most of the successful OSS groups seldom develop in isolation; and their bugs and the underlying debugging processes are likely to be intertwined among multiple groups. We argue that the interrelatedness among software groups is an outcome of the long range contacts established through the boundary spanning activities of their contributors. Long-range contacts can exert an inverted U-shaped impact on releases, that is, initially, as conduits of valuable information, they benefit releases. But too many contacts slow down releases. We also hypothesized the influence of long-range contacts on releases is moderated by the relative location of OSS groups. We tested these intricate relationships using the contributions made by 7078 developers in solving over a million of highest priority bugs from 2343 software groups. Our empirical models are largely supported.

Keywords: Long-range contacts, resource heterogeneity, k-core, clustering coefficients

1 INTRODUCTION

Often OSS and its proprietary counterparts are contrasted in parallel with imageries of the cathedral and the bazaar. Specifically, the distinctiveness of OSS is uniquely depicted in the Linus' Law, “given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.” This is further to another principle that "release early and release often". Yet their intuitive appeals remain largely anecdotal, and have been recently contested to have a detrimental effect due to unfocused contribution. For example, in assessing the impact of developer activities on the known security vulnerabilities in the Red Hat Enterprise Linux 4 kernel, Meneely and Williams (2009) found that too many independent contributors can dilute the focus and lead to submission of vulnerable software patches. The deleterious effect underscores the mantra of not having "too many cooks in the kitchen". This follows Brook's law that too many contributors inevitably increase coordination and communication costs. In this paper, we conduct further empirical testing of these two principles using a much larger population of OSS groups.

Our premise is that the realized benefits of having contributors with similar roles in doing the same thing are lesser than having multiple roles involved in debugging. The rationales include: having many contributors performing the same role may lead to duplication of efforts and wastage, assuming that voluntary contribution holds; and unlike most of the ill-structured tasks, debugging can often be done individually although collective action can speed up the process (Scacchi 2004). Whereas having multiple roles can bring forth the needed complementarities between users and developers as each role presents a unique set of resources to the debugging process. This distinction underlines our first contribution by explicating the essence of eyeball principle to resource heterogeneity, and its influence on software release. We choose software release as it is closely related to how efficient bugs are fixed, i.e. the quicker the bugs are fixed, the more often the releases. The second contribution addresses a neglected aspect of OSS development in relation to the boundary spanning activities of the users and developers across different OSS groups. The boundary spanners serve as conduits of valuable information among otherwise isolated OSS through an increase in long range contacts. Users can also play multiple roles in their affiliations, that is, a developer can be a user in one group and a core developer in another. This increases resource heterogeneity as each role presents a unique set of resources to debugging. Yet increased affiliations and contacts can lead to diluted attention and effort. This underlines our third contribution in identifying and ascertaining the limits of the eyeball and the release principles.

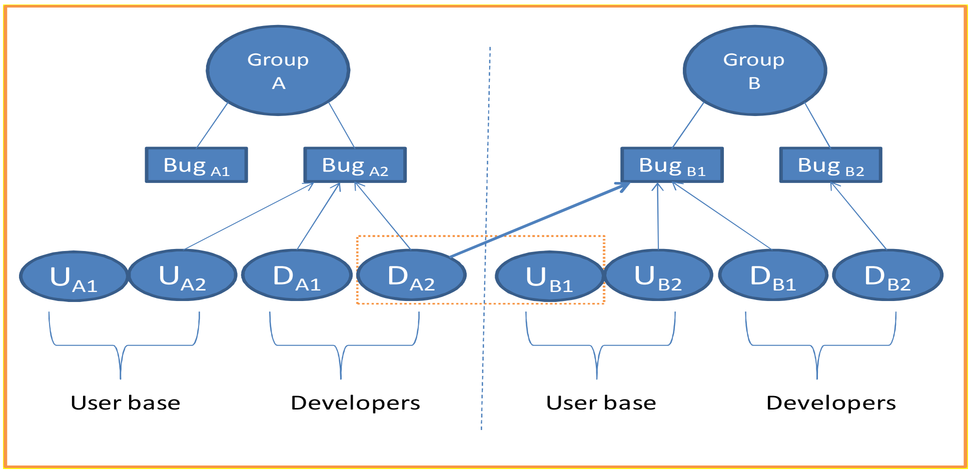
1.1 Theory and hypotheses

In contrast to proprietary software, the bazaar model of OSS development underscores participation of contributors is voluntary[1]. That is, a developer can choose which OSS group to contribute. Bug fixing presents an opportunity that every contributor including users and developers can contribute. Users can report bugs, and submit patches; and users that do not write software will have to rely upon others. The eyeball principle underlines the complementary relationship between users and developers. Yet it is not clear what constitutes "enough eyeballs", and "whose eyeballs" are more valuable to debugging. Also consider there are many software groups, individual contributors can freely join as many groups as they like although contributors tends to gravitate towards a few OSS groups (Maddley et al. 2004). That aside, the affiliation of each contributor presents an unique set of resources. This follows the strength of weak ties (Granovetter 1973), such that affiliations can be conceptualized as long-range contacts which serve as strong conduits of valuable information.

[1] Here in our study, we have no means of ascertaining the motivation of contributors, and differentiating developers into pure hobbyists and corporate sponsored ones.

Figure 1 shows two software groups A and B. They are related via the boundary spanning activity of developer D_{A2}. Software group A has its own user base (including U_{A1} and U_{A2}) and developers (D_{A1} and D_{A2}). The developer D_{A2} not only contributes to the debugging of Bug_{A1} but also to Bug_{B1} in the capacity of a user rather than a developer.[2] The affiliation via boundary spanning activity of D_{A2} can affect software group A in two unique ways: first, software group A may no longer develop and maintain in isolation but in tandem with group B; and second, affiliation has made unique knowledge specific to software groups A and B more fluid and accessible to members of group A.

Figure 1. An illustrated example of the relationship between affiliation and software relatedness



The first effect of affiliation increases software relatedness. Software groups are related when they share common contributors. The concept of relatedness is not new in the literature and is said to increase knowledge flow and exploitation through reuse (Tanriverdi 2005). Software groups that are related through affiliation are likely to be aware of the latest development of each other, and possibly schedule and work in tandem. For example, a contributor may contribute to the code libraries of a scripting language, and have the chance of interacting with other contributors. Because of this contribution and interaction opportunity, the contributor may acquire new technological insights into how to use his or her contributed code and combine with other codes in developing newer and better software applications. This increase in software complementarities accelerates software releases.

The affiliations and interactions among contributors across different software groups also increase the awareness of who are in the neighbourhoods and who the experts are when it comes to certain bugs. This increase in awareness reinforces the notion of a small world network commonly observed in social systems (Kossinets and Watts 2006). A small-world network is said to comprise a number of cohesive groups or connected components via bridges or 'short-cuts' (Watts et al. 1998). In relation to Figure 1, the debugging contribution by developer D_{A2} to fix Bug_{B1} provides an interaction opportunity with U_{B2} and D_{B1}. The interaction link creates a short-cut between software groups A and B. The interaction link via D_{A2} forms the long-range contacts of D_{A1} and U_{A2}. In view of the affiliation dynamics, we characterize software relatedness as the long-range contacts, measured as the degree of clustering of users and developers of a software group within their immediate neighbourhoods.

These long range contacts can benefit collaborative activities in bug fixing as they will increase awareness of requirements, knowledge transfer, and software complementarities across different OSS groups. Yet too many contacts not only increase the coordination cost which slows down bug fixing but also impose constraints on design choice (Nickerson 2004). This inevitably slows down cycles of software development. That is, the small world benefits associated with the long-range contacts

[2] Contributors acquired the status of a developer if they have the access right to write and modify the code library of the software group otherwise they assume the status of a user.

attributed to the boundary spanning activities of contributor will only remain positive up to a certain limit. We formulate the following hypothesis to ascertain the influence of long range contacts on software releases.

> *Hypothesis 1: Long range contacts exhibit curvilinear relations with software releases, first rising and then declining (an inverted U-shape), reflecting the fact that long contacts exert a negative effect on group releases at its extremes.*

Although contributing to debugging is entirely voluntary, whether individual contribution will be accepted by others is a different matter. Lave and Wenger (1991) refer the initial contribution to legitimate peripheral participation through which newcomers become socially accepted and included in a community of practice. In order for the newly joined OSS developers to increase the chance that their contributions will be accepted and reciprocated, they have to embark on a social learning process by climbing some sort of apprenticeship ladder (Kim 2000), and by following some form of joining script specific to an OSS project (see e.g., von Krogh et al. 2003) or both. Ye and Kishida (2003) suggest there are at last eight different roles based upon their contributions in software development. In relation to debugging, the eyeball principle suggests beta-testers (or users) and co-developers. We contend that the boundary-spanning developers contribute another set of resources as they provide the conduits of valuable information and knowledge. Although there are status differences among them, they all contribute in their own unique ways. Table 1 summarizes how each role (as depicted in Figure 1) contributes to resource heterogeneity that matters to debugging.

Table 1. Membership resource heterogeneity

U_{A2}: User	D_{A1}: Core Developer	D_{A2}: Core and peripheral
A user/beta-tester Role: testing, bug reporting, sometimes contributing patches of code and forming part of the user-base	A developer with CVS account Role: knowledge brokering, writing the majority of software codes and bug fixing	A developer who is core to group A but serves as a user to group B. Role: boundary spanning and knowledge brokering

We contend that all three roles are equally important. The users not only contribute in the capacity of beta testers but also their bug reporting often incentivizes developers because developers often revealed that they feel satisfied from helping the users (Wu et al. 2007). Although contributors that criss-cross different software groups are more likely to provide the conduits of valuable information and knowledge, other contributors that criss-cross less but dedicate more time to fixing bugs are equally invaluable to debugging. Hence, software groups comprising members of performing diverse roles (user; core; and core and peripheral) are likely to benefit from the unique sets of resources that each specific role brings. The above consideration has led to the following hypothesis:

> *Hypothesis 2: Resource heterogeneity will increase software releases*

Yet the small world benefits (attributed to long-range contacts) and the resource heterogeneity (as a result of diverse membership roles) are likely to be affected by the relative location among software groups. In contrast to isolated software groups, software groups that are well connected are likely to attract the attention of the majority of the OSS developers (Maddey et al. 2004). This follows the notion of preferential attachment, or the so-called Matthew effect. In relation to debugging, solving a bug of a well connected software group has more appeal as it will induce more satisfaction and have a wider impact on a larger user-base than solving a bug of a less well connected group of a smaller user-base. However, working with too many developers will lead to unfocused contribution (Meneely and

Williams 2008) and increase the coordination and communication costs (Brooks 1995), and the depth of discussion and knowledge sharing will remain shallow (Benbunan-Fich et al. 2002). Software groups that are located within a denser core are likely to dilute the benefits of long range contacts and resource heterogeneity. This has led to the following two hypotheses:

Hypothesis 3a. Location will moderate the impact of long range contacts on group releases

Hypothesis 3b. Location will moderate the impact of resource heterogeneity on group releases

2 RESEARCH METHODS

We used SourceForge.net (SourceForge) as our data source as it is generally considered to be the largest repository of OSS groups. Also with the free tools provided by SourceForge, such as concurrent versioning systems (CVS), mailing lists and bug tracking systems (tracker), we were able to extract behavioural information of over a million contributors from the relevant archives. This approach has proven to be rigorous in studying user preference, collaboration amongst developers, and maintenance activities (e.g. Antoniades et al. 2007). Data logs from the tracker systems were used to analyse how users and developers contributed to debugging. The tracker allows developers and users to report and manage bugs, patch submissions, support requests and feature requests. Each project may have multiple trackers. That is, each project may have many logged bug requests. The tracker provides information such as the summary of a request, open date, close date, update date, bug status (e.g. open, close), resolution status (e.g. fixed, duplicate, none, rejected, etc.), submitter ID, and assignee ID.

Every month, a complete dump of the SourceForge database is shared with Notre Dame (minus personal information and OSTG specific and site functionality specific information for security and privacy reasons). In other words, a snapshot of all activities from the beginning until the date of the dump is available. For the purpose of our study, the latest dump that we used was June 2009. Thus, data used for analysis contains all bug activities from the start of data recording until June 2009. Before we extracted the data from the Notre Dame database, we studied how it was organized and structured in a format of an Entity Relationship (ER) diagram. Then we wrote Standard Query Language (SQL), to retrieve the information from the database.

Four types of bugs are listed in SourceForge. They are support requests, bugs, feature requests, and patch submission. Only bugs (defects) were used for our study because they clearly show corrective and perfective change. The others are less obvious since some of them are minor questions and duplicate of reported bugs. Each group may contain hundreds to thousand numbers of bugs. To limit the number of bugs examined, we selected bugs with the highest priority determined by the elders of the OSS groups. This resulted in 1,073,256 bugs from 2343 OSS groups with 7078 contributors.

2.1 Measures

*Release*s. We used software releases as our dependent variable considering our focus on examining the relationship between the eyeball and the release principles. We counted the total number of versions of each software group, mean = 17. 31, sd = 27.54.

Long range contacts. We calculated long range contacts of a software group by averaging the individual clustering coefficients, which are the proportion of ties that exist among the collaborators of a developer against the potential number of ties that could exists within and across OSS groups.

Resource heterogeneity. We used the Gini index of diversity to measure the distribution across three distinctive membership roles comprising 'user', 'core developers', and ‘core and peripheral developers'. The index is defined as, $G = 1 - \sum p_i^2$. At maximum homogeneity, the index is zero, whereas at maximum heterogeneity, it has the value of (1 - 1/n).

Location. The relative locations of OSS groups were measured using the k-core algorithm in Pajek. A k-core is defined as a maximal subnetwork in which each node has at least degree k within the subnetwork (de Nooy et al. 2008). For example, if a k-core has a value of 4, it indicates that all nodes (or vertices) has at least 4 other nodes within the subnetwork. Our results indicated three distinctive groupings. The first k-core (kcore1) constituted 75% of the total software groups and contained all the isolated OSS groups with k = 1. We labelled this grouping as 'periphery location'. The second k-core (kcore2) constituted 17.2% of the total observation with k values ranging from 3 to 6. We labelled this grouping as 'semi-periphery location'. The third, remaining k-core grouping (7.8%) had the largest k value of 210. We labelled this as 'core location' (kcore3). The relative locations are included in the Appendix.

Control variables. We used two control variables. First control is the software age. To standardise the software age across projects, June 2009 was chosen as the anchor date. The software age is then calculated by subtracting the anchor date with the registered date of the group. The final time is recorded in years, mean = 5.15, sd = 2.34. Second control is the number of contributors. As debugging was open to all developers, the number of developers would increase the bug fixing and in turn increase releases. Hence, the total number of developers that contributed to bug fixing of each software group was included as a control variable.

2.2 Data analysis

The affiliations between contributors and software groups can be modelled as a two-mode undirected network (Wasserman and Faust 1994). A node *d* representing a contributor is associated with another node *g* representing a software group when *d* is part of *g*'s team of contributors. To calculate individual clustering coefficient of *d*, we transformed the two-mode network into a single-mode network with all nodes representing contributors. Two nodes, *d*1 and *d*2 are connected if they both contributed to a bug of a same software group. All the transformation and calculation were carried out in Pajek. Since our dependent variable, software release, was a count variable and took on non-negative integer values, we used the negative binomial regression approach to avoid heteroskedastic, nonnormal residuals (see Hausman et al. 1984). To test the curvilinear effect of hypothesis 1, we first mean centered long range contacts, before squaring. This is to reduce the potential problem of multicollinearity (Aiken and West 1991). In the regression models, we entered both linear and squared terms, the beta coefficient of linear term has to be positive, and notably the squared term has to be negative and significant.

Table 2. Descriptive Statistics and Correlations

Variable	Mean	Std. Dev.	1	2	3	4	5
1 Release			1.00				
2 Age (Year)	17.31	27.54	0.13 ***	1.00			
3 Number of Contributors	5.15	2.34	0.04	0.06 **	1.00		
4 Long Range Contacts	3.00	25.82	0.04 $^{+}$	-0.05 *	0.08 ***	1.00	
5 Heterogeneity	0.05	0.13	0.21 ***	0.09 ***	0.09 ***	0.08 ***	1.00
6 Long Range Contacts Squared	0.16	0.23	-0.11 ***	-0.04 *	-0.04	0.04	-0.19 ***

n = 2343; $^{+}$ p < .08* p < .05, ** p < .01, *** p < .001.

3 RESULTS

3.1 Descriptive Statistics

Table 2 reports descriptive statistics and inter-correlations for all the variables of interest. Releases had a standard deviation greater than its mean, which indicated overdispersion. We found releases to be positively and significantly correlated with the total number of contributors and heterogeneity, but the correlation between releases and long-range contacts was marginally significant. The long-range contacts squared term was negatively and significantly correlated with releases.

3.2 Regression Models

We report the results of the negative binominal regression analysis in Table 3. Model 1 is the unconstrained controls-only model. Model 2 includes long range contacts as linear and quadratic terms to test hypothesis 1, and heterogeneity for hypothesis 2. Model 3 introduces the interaction effects to test hypotheses 3a and 3b.

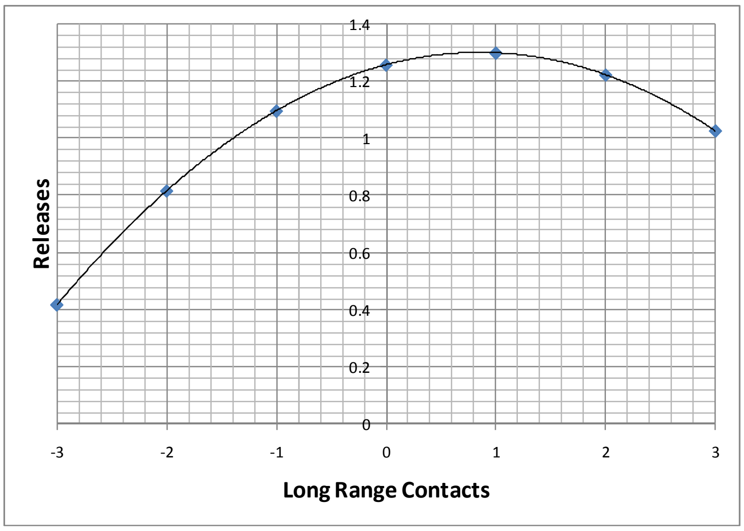
Table 3. Negative Binominal Models with Fixed Effects for Software Releases[a]

Variable	Model 1			Model 2			Model 3		
Constant	2.33	***	(0.05)	2.45	***	(0.07)	2.41	***	(0.08)
Independent									
Long range contacts				0.25	+	(0.17)	1.16	***	(0.36)
Heterogeneity				1.13	***	(0.10)	1.04	***	(0.12)
Long range contacts squared				-3.35	***	(0.56)	-3.47	***	(0.74)
Moderating									
Kcore2 (Semiperiphery vs. Periphery)[b]							0.15		(0.08)
Kcore3 (Core vs. Periphery)[b]							0.5	**	(0.20)
Kcore2 X Long-range Contacts							-1.84	***	(0.47)
Kcore3 X Long-range Contacts							-1.39	**	(0.56)
Kcore2 X Heterogeneity							0.48	*	(0.24)
Kcore3 X Heterogeneity							-0.91	*	(0.37)
Control									
Age (year)	0.08	***	(0.01)	0.07	***	(0.01)	0.07	***	(0.01)
Number of contributors	0.02		(0.00)	0		(0.00)	0		(0.00)
df	2			5			11		
Log-likelihood	-9014.15			-8923.36			-8905.96		
Log-likelihood ratio				181.58			216.38		
Wald c2	131.26	***		150.05	***		195.56	***	

[a]n = 2343; standard errors are shown in parenthesis
[b] dummy coded
$^+$ p < .08
* p < .05
** p < .01
** p < .001

Hypothesis 1 posits an inverted U-shaped relationship between releases and long range contacts. The results in Model 2 indicate that the linear term is positive and marginally significant (p< .08), and the squared term of long range contacts is negative and significant (p < .001), thus supporting hypothesis 1. Figure 2 (plotted using model 2, ranging from -3sd to +3sd) demonstrates the overall effects of long range contacts on releases. The figure clearly shows the inverted U-shaped between releases and long range contacts, and slope analysis (Greene 1997) confirms that positive slope apparent at the lower end and releases turns negative at higher end of long range contacts.

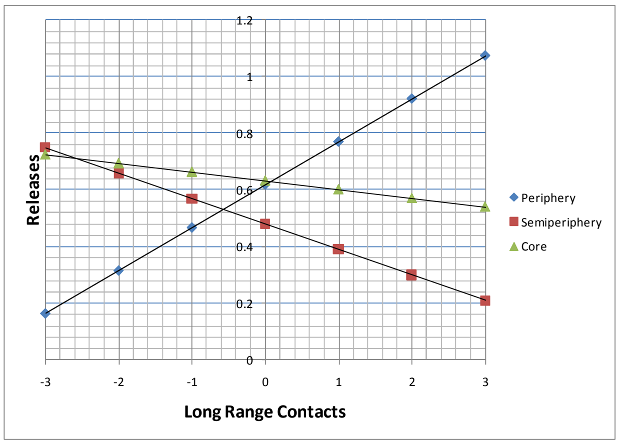
Figure 2. Curvilinear Effect of Long Range Contacts on Software Releases



3.3 Moderating effects

To test for the moderating effects, we created linear interaction terms composed of long range contacts and heterogeneity with each of two moderating locations (core vs. periphery; and semi-periphery vs. periphery). We entered the interaction terms as a block in Model 3 to account for their simultaneous effect on software releases. Evidence of moderation is found and the model fits well (Golden and Viega 2005). To better interpret the interaction terms, we graphed the interaction effects (only the interaction effect between long range contacts and locations was shown in Figure 3) following procedure outlined in Cohen et al. (2003). Figure 3 show that in contrast with software groups located at the core and semi-periphery, an increase in heterogeneity will positively increase software releases of groups located at the periphery.

Figure 3. Interaction Effect of Relative Location of Software Groups



3.4 Robustness checks

To check for robustness, we used robust standard errors to estimate the significance levels of the main, curvilinear and moderating effects. The interaction effects comprising of heterogeneity and the two moderating locations were not significant. But the linear and quadratic effects of long-range contacts remain significantly. We also used another set of data (based on a data dump of September 2008) to replicate the above empirical models, they were all supported including the interaction effects of heterogeneity. Overall, the results provide strong evidence for the curvilinear effects of long-range contacts, the linear effect of heterogeneity on releases, and the moderating effects of location on the relationship between long range contacts and releases.

4 DISCUSSION

The primary aim of this study is to understand the eyeball principle relating to how it works, and how it affects software release. This study makes several contributions to the extant literature of OSS participation. First, it builds upon the unfocused contribution argument by identifying further set of conditions that may be detrimental to software outputs. Although the previous research has underlined the demise of having too many independent contributors, we argue that even when contributors are interdependently and collaboratively engaging in debugging, too many of them will still tip the balance and slow down software releases. This is directly opposite to unfocused contribution. It seems that increased software relatedness may make contributors even more focused and mindful of other groups' requirements, and possibly the wider impacts and repercussions of their contributions to other OSS groups. This is increasingly the case as bug fixings and new feature implementation have become more forward than backward compatible (HDF 2009). Although we do not test this in the present study, future research can examine whether a drop in group releases actually benefits the quality of the software groups, and prolongs the longevity of the software groups.

Second, it moves beyond a single case of empirical analysis to a larger population of OSS groups. This is important to better understand and capture the participation dynamics of contributors and the intricate relationships among OSS groups. By studying the debugging of the highest priority bugs, it is likely that we have captured a very high proportion of competent OSS developers. The findings are not trivial, as the results (Model 1) indicate that software outputs are not determined by how many contributors involved in debugging but what really matters is the long range contacts that each contributor brings to debugging, and that too many will only slow down group releases.

Third, we have ascertained the significance of contributors that criss-cross OSS groups in addition to the normal distinction between core and peripheral members. Contributors of dual roles (core and peripheral) are likely to serve in the capacity of a technical gatekeeper (Allen 1977) and engage in a balancing act between boundary spanning and knowledge brokering. We further argue that their significant contribution is through establishing long range contacts for other members of the groups. Unlike most of the closed networks commonly found in firms and social networks, the openness and transparency in open source networks have made structural holes redundant as debugging and OSS development is open to everyone to participate. Use the example given in Figure 1 to provide a more succinct illustration. The interaction link established through D_{A2} provides a long-range contact for D_{A1} and U_{A2}. Although D_{A2} is occupying the structural hole between software groups A and B, there is no stopping for both D_{A1} and U_{A2} contributing to the debugging of Bug_{B1}. Yet it is unlikely as it may duplicate the effort, and it is also possible that both D_{A1} and U_{A2} have their own long range contacts to maintain. Future research can examine this further, by monitoring how individual networks of long range contacts unfold over time and whether they remain stable.

As with any study, there are limitations that must be evaluated. First, only the highest priority bugs are examined. The findings may be different in low priority bugs as highest priority provided a stronger signal to the contributors to involve. This needs further research. Second, in operationalizing long range contacts and location, we draw on the established measures from the small world literature (Watts et al. 1998) and social network analysis (Nooy et al. 2008). It is possible other equally valid measures may exist. Future research can compare the present set of measures with other measures to allow further testing of reliability and validity. Third, although we used another set of data dump to replicate our findings, panel data can be collocated and provide better testing of the causal relationships as hypothesized in the present study.

References

Aiken, L. S. and West, S. G. (1991). Multiple regression: Testing and interpreting interactions. Thousand Oaks, CA: Sage.

Allen, T. J. (1977). Managing the flow of technology ; technology transfer and the dissemination of technological information within the R&D organization. Cambridge, Massachusetts.: MIT Press.

Antoniades, I., Samoladas, I., Sowe, S.K., Koch, S., Fraczek, K. & Hadzisalihovic, A. (2007) "Free/Libre and Open Source Software Metrics", February 15, FLOSSMetrics Consortium 2006-2007

Benbunan-Fich, R., Hiltz, S. R. and Turoff, M. (2002). A comparative content analysis of face-to-face vs. asynchronous group decision making. Decision Support Systems, 34(4) 457–469.

Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Reading, MA.

Cohen, J., Cohen, P., West, S. and Aiken, L. (2003). Applied multiple regression/correlation analysis for the behavioural sciences (3rd ed.). Hillsdale, NJ: Erlbaum.

de Nooy, W., Mrvar, A. and Batagelj, V. (2008). Exploratory social network analysis with Pajek. Cambridge, UK: Cambridge University Press

Golden, T. D., and Viega, J. F. (2005). The impact of extent of telecommuting on job satisfaction: Resolving inconsistent findings. Journal of Management, 31(2), 1-15.

Granovetter, M. (1973). The strength of weak ties. American Journal of Sociology, 78 (6), 1360-1380.

Greene, W. H. (1997). Econometric analysis (3rd ed.). Upper Saddle River, NJ: Prentice-Hall.

Hausman, J., Hall, B. and Griliches, Z. (1984). Econometric models for count data with an application to the patents-R&D relationship. Econometrica, 52, 909-938,

Kim, A. J. 2000. Community building on the Web: Secret strategies for successful online communities. Addison Wesley, London, UK.

Kossinets, G. and Watts, D. J. (2006). Empirical analysis of an evolving social network. Science, 311, 88-90.

Kuk, G. (2006). Strategic interaction and knowledge sharing in the KDE developer mailing list. Management Science, 52(7), 1031 - 1042.

Lave, J., E. Wenger. 1991. Situated learning. Legitimate peripheral participation. Cambridge University Press, Cambridge, UK.

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.

Meneely, A. and Williams, L. (2009). Secure open source collaboration: An empirical study of Linus' law. Paper accepted at the 16th ACM Conference on Computer and Communications Security, Chicago, November. http://www4.ncsu.edu/~apmeneel/ccs221-meneely.pdf

Nickerson, J. A., T. R. Zenger. 2004. A knowledge-based theory of the firm: The problem-solving perspective. Organization Science, 15(6), 617–632.

Oregon, USA.

Scacchi, W. (2004). Free and open source development practices in the game community. IEEE Software, 21, 56-66.

Tanriverdi, H. (2005). Information technology relatedness, knowledge management capability and performance of multibusiness Firms. MIS Quarterly, 29 (2), 311-334.

The HDF group. Backward and forward compatability. September 1, 2009. www.hdfgroup.org/HDF5/faq/bkfwd-compat.html.

von Krogh, G., S. Spaeth, K. R. Lakhani. (2003). Community, joining, and specialization in open source software innovation: A case study. Research Policy, 32,1217–1241.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. Nature, 393, 440-442.

Wu, C. G., Gerlach, J. H. and Young, C. E. (2007). An empirical analysis of open source software developers' motivations and continuance intentions. Information & Management, 44, 253-62.

Ye, Y. and Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. In Proceedings of the 2003 International Conference on Software Engineering, Portland.

5 APPENDIX.

The three tier structure comprised core, semi-periphery and periphery locations. The network layout was generated using the Kamada Kawi free energy layout algorithm in Pajek.