Core and periphery in Free/Libre and Open Source software team communications

Kevin Crowston, Kangning Wei, Qing Li & James Howison
Syracuse University School of Information Studies
crowston@syr.edu, kwei@syr.edu, qli03@syr.edu, jhowison@syr.edu

Abstract

The concept of the core group of developers is important and often discussed in empirical studies of FLOSS projects. This paper examines the question, "how does one empirically distinguish the core?" Being able to identify the core members of a FLOSS development project is important because many of the processes necessary for successful projects likely involve core members differently than peripheral members, so analyses that mix the two groups will likely yield invalid results.

We compare 3 analysis approaches to identify the core: the named list of developers, a Bradford's law analysis that takes as the core the most frequent contributors and a social network analysis of the interaction pattern that identifies the core in a core-and-periphery structure. We apply these measures to the interactions around bug fixing for 116 SourceForge projects. The 3 techniques identify different individuals as core members; examination of which individuals are identified leads to suggestions for refining the measures. All 3 measures though suggest that the core of FLOSS projects is a small fraction of the total number of contributors.

1. Introduction

The concept of a core group of developers is an important one, often discussed in empirical studies of FLOSS projects. This paper examines the question of how researchers can empirically distinguish members of the core group and provides some evidence regarding the size and composition of the core group for a sample of FLOSS projects.

Academic case studies of FLOSS projects [e.g., 4, 6, 7, 11-13] suggest that FLOSS development teams have a hierarchical structure. For example, Mockus et al. [12] studied the Apache `httpd` project and found that development was quite centralized, with only about 15 developers contributing more than 80 percent of the code for new functionality. Bug reporting, on the other hand, was quite decentralized, with the top 15 reporters submitting only 5 percent of problem reports in the Apache project. They summarize this finding by hypothesizing that, "In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems." (p. 329). Moon and Sproull [13] in a case study of the development of Linux similarly describe a highly skewed distribution of traffic on the Linux mailing lists.

The suggested hierarchical or onion-like structure is shown in Figure 1. At the center of the onion are the core developers, who contribute most of the code and oversee the design and evolution of the project. In the next ring out are the co-developers who submit patches (e.g., bug fixes), which are reviewed and checked in by core developers. Further out are the active users who do not contribute code but provide use cases and bug reports as well as testing new releases. Further out still, and with a virtually unknowable boundary, are the passive users of the software who do not contribute to the project's lists or fora. In this paper, we focus on distinguishing the core developers (the core) from co-developers and active users (grouped together as the periphery); we do not consider passive users.

Identifying the core group of developers is an important question for empirical research for several reasons. First, while the hypothesized team structure shown in Figure 1 has a great deal of face validity, it has not yet been fully tested on a range of projects. In part this is because it is not clear how to operationalize the intuitively appealing notion of a core group. Second, being able to identify the core members of a FLOSS development project is important because many of the processes necessary for successful projects (e.g., development of shared un-

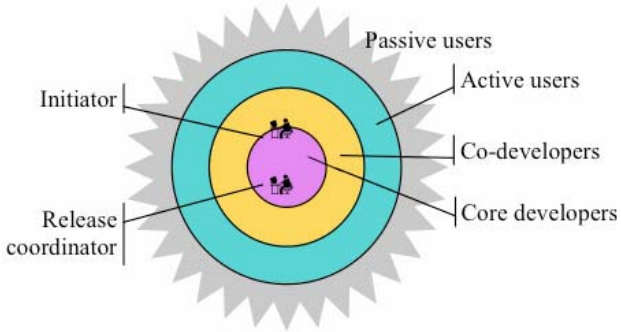


Figure 1. Hierarchical model of FLOSS development team structure.

derstandings of user needs or system architecture, conflict resolution, leadership, etc.) likely involve core members differently than peripheral members. Including all developers in an analysis without distinguishing their roles would likely lead to equivocal results. Finally, being able to separately count the core and peripheral developers should provide some insight into the state of development of the team in accordance with Mockus et al.'s [12] hypothesis.

As well, our study of core members may be of more general relevance. Prior research suggests that small groups have a typical size. For example, James [9] reported that action-taking groups in a variety of settings averaged only about 5 to 7 members and observed free-forming groups were even smaller. He argued that because of the cost of maintaining relationships with other group members, free forming groups will have an average size of only 3, with a range of 2–7. Because this limit is due to information processing constraints, it is plausible that groups interacting primarily with Computer Mediated Communication might be able to support a higher level of interaction and thus larger groups. Studies of distributed teams, such as FLOSS development projects, will provide evidence on this point.

Therefore, the research question addressed in this paper is how to empirically identify the core members of FLOSS teams. The secondary research question is to understand the size and composition of the core groups of a population of projects and to see how these vary from project to project. Our analysis will provide evidence for the generalizability of the hypothesized core-and-periphery structure.

2. Methods and data

We identified three approaches to identify the core group, from simple to complex: self-report on project home pages, level of contribution, and core-and-periphery social network analysis.

- First, and most simply, the core may be defined as those individuals who are officially named as developers on the project and the periphery all other contributors.
- Second, the core may be defined as those who contribute the most to the project. Most projects demonstrate a very skewed distribution of levels of contribution: a few individuals contribute a lot while most contribute only a little. Therefore, the core group are the few members that contribute the most while the others are the periphery.
- Finally, the core may be defined from the pattern of interactions. According to Borgatti and Everett (1999), a core-and-periphery network "entails a dense, cohesive core and a sparse, unconnected periphery" (p. 375). Therefore, if this structure applies, team members can be partitioned into two groups, with the core defined as the tightly interconnected group and the periphery as the disconnected group.

If the three analysis approaches identify more-or-less the same members as being members of the core, then future research can use the simplest approach. On the other hand, if the methods yield different answers, then it will be necessary for researchers to pick the analysis technique that is most appropriate for their research question.

2.1. Data

To study the contribution of developers, we collected demographic and interaction data from projects. The specific demographic data we collected were the lists of developers and their roles in projects. To measure contributions, we considered analyzing code, developer mailing lists and bug trackers. We found that code could not be reliably attributed to particular authors due to differences in the use of source code control systems in different projects. As well, in most projects, only listed developers can check in code, so contributions from active users are not visible. Finally, we wanted to compare an analysis based on the level of contributions to one based on interactions, which are not represented in code.

In comparing the possibility for analyzing interaction data, we considered developer mailing lists and bug trackers. In the end, we chose to analyze data from bug trackers for both contributions and interactions. We chose this data source for several reasons. First, bug fixing is a collaborative task in which, as Raymond [14] paraphrases Linus Torvalds, the people finding the bugs are different from those that understand the bug and those that fix the bug. (Indeed, this approach to bug fixing is the basis for some claims of effectiveness made for the FLOSS development approach.) As such, bug fixing provides a "microcosm of coordination problems" [5]. Second, we found that we had data from more bug trackers about more projects. Finally, as mentioned above, Mockus et al. [12] found that bug reporting involves the broadest range of project participants. Thus the collaboration involved in bug fixing produces rich data about interactions that involve the entire community, the core and co-developers as well as active users, and thus provides evidence regarding the social structure of the entire membership of the development teams. For these reasons, we based the analysis in this paper on data from the project bug tracking systems.

Unlike Mockus et al. [12], we examined the entire interaction around a bug report, not just the initial report. Contribution and interaction data were obtained by examining the level and pattern of messages posted to the bug trackers. In addition to tracking the status of bugs, bug-tracking systems enable users to report, and developers to discuss bugs (though projects vary in how they use these systems). As shown in Figure 2, a bug report includes basic information about the bug that can be followed up

with additional messages seeking or providing additional information about the bug. We analyzed these follow-up messages for evidence about the contribution of developers and the resulting social structure of the teams.

2.2. Sample of FLOSS projects

The sample of projects we analyzed was drawn from projects hosted by SourceForge (http://www.sourceforge.net/), a free[1] Web-based system that provides a range of tools to facilitate FLOSS development. At the time of our data collection, SourceForge supported more than 50,000 FLOSS projects on a wide diversity of topics[2]. Clearly not all of these projects were suitable for our study: many are inactive, previous studies have suggested that many are in fact individual projects [10], and some do not make bug reports available. Therefore, we restricted our sample to projects that listed more than 7 developers and had more than 100 bugs in the bug tracking system at the time of selection in April 2002. We identified only 140 projects that met these criteria. Though it was not an explicit goal of the sampling, most if not all of the projects selected are likely to be successful, in that they have succeeded in attracting developers and attention from users, as reflected in the bug reports.

2.3. Data collection

To collect data, we developed programs to download and parse the bug report pages for the selected projects. Bug report pages were spidered from SourceForge in April 2003. Unfortunately, between selection of projects and data collection, some projects restricted access to bug reports, so we were able to collect data for only 122 projects. Of these, a further 6 had few distinct posters to the bug tracker, resulting in a sample of 116 projects for analysis. The list of developers per project was obtained from the OSSMole project [8].

Processing the SourceForge data revealed a problem with missing data. Specifically, when a message is posted by a non-logged-in user, the sender is listed as "nobody". These messages constituted an average of 15% of the messages (as low as 0% and as high as 50% for a few projects). We considered several alternative strategies for handling this missing data and decided to recode the "nobodies" as a unique individual in each bug report (e.g., using "nobody686314" as the sender of all "nobody" messages in bug report number 686314). This approach retains interactions between individuals but at the cost of introducing of fictitious characters. Table 1 lists examples of the projects to give a sense of the sample. Those familiar with FLOSS may recognize some of these projects, which span a wide range of topics and programming languages.

Table 1. Examples of projects included in sample.

Project name	Short description
Curl	Command line tool and library for client-side URL transfers.
gaim	A GTK2-based instant messaging client.
netatalk	A kernel-level implementation of the AppleTalk Protocol Suite.
phpmyadmin	Handles the basic administration of MySQL over the WWW
squirrelmail	A PHP4 Web-based email reader.
Tcl	Tool Command Language

3. Analysis

In this section, we provide the details of the three different analyses we used to identify the core members of FLOSS teams.

3.1. Formal roles

The first analysis relied on self-reported formal roles, as shown in the list of developers from the project Web pages. Unfortunately, the list of developers was collected at a different time than the interaction data (October 2004). Since most projects grant developer status based on a track record of contributions, we might identify individuals as core based on their contributions before they were formally recognized as such. Having a gap between the two data collection points slightly increases the chance of this mismatch between the analyses. OSSMole also includes information about the project administrators and specific roles, but we did not use these for this analysis.

3.2. Distribution of contributions

The second analysis is based on the numbers of postings from different developers. As mentioned above, the distribution of numbers of postings is heavily skewed: a few developers post many messages, while most post only a few. We based our analysis on Bradford's Law of Scatter [2]. Working in the area of bibliometrics, Bradford [2] found that when compiling a bibliography on a particular subject, a few journals would have many articles on the topic, but a few articles could be found in many journals. Empirically, the count of articles per journal followed a characteristic skewed distribution, namely, the Bradford or Zipf distribution. The ubiquity of this distribution has led Brookes [3] to propose it as a universal law of human

[1] At least free 'as in beer': ironically, the SourceForge system itself is now proprietary. Savannah was developed by the Free Software Foundation's GNU project from the last free 'as in freedom' version of SourceForge.

[2] As of 15 June 2005, SourceForge claims 101,571 projects.

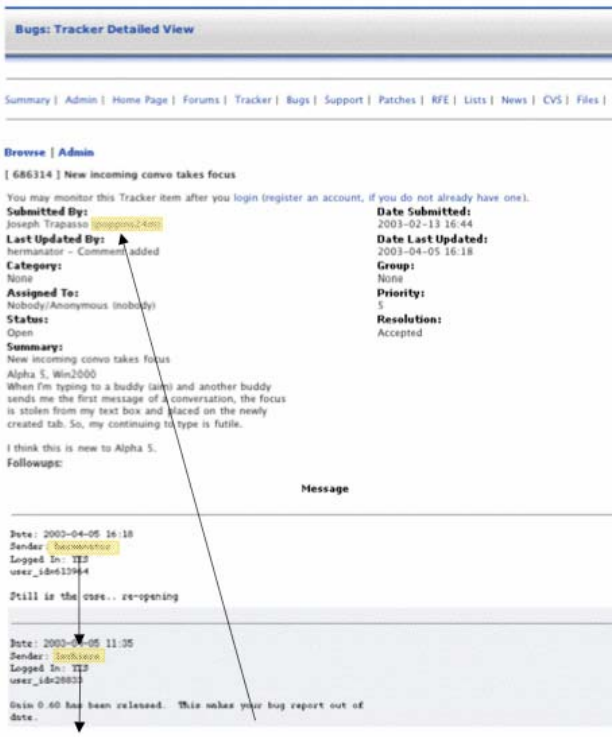


Figure 2: Example SourceForge bug report and follow up messages showing coding of interactions (http://sourceforge.net/tracker/index.php?func=detail&aid=686314&group_id=235&atid=100235

behaviour describing any situation where success leads to further success, prompting our application of the law to this situation. Based on the mathematical properties of this distribution, Bradford defined the core set of journals on a topic as those that contribute 1/3 of the total number of articles. By analogy, we define the core group as the members that contribute 1/3 of the total number of posting to the discussion.

For this analysis, we first determined the frequency of postings for each team member. In the bug tracker systems, contributions to the system (bug reports and follow up messages) are identified by a unique user ID, which we used to identify members. It is possible (and there is no way of knowing) that a single individual could utilize multiple IDs or that multiple individuals could share one. However, we believe it is unlikely that many do either due to the logistics of maintaining multiple accounts and the lack of incentive to do so. Indeed, because reputation accrues to an ID, we believe that most individuals will choose to maintain a single ID. We then listed members in descending order of contributions. We take as the core group the team members who contributed the most and whose total contributions equal (or exceed) 1/3 of the total (though as noted below, because of the extreme skewness of the contributions, we also examined the group that contributed 2/3 or more of the postings). In the event that two members are tied in frequency, they are placed together in the same group.

3.3. Core and periphery

The final analysis was based on a core-and-periphery analysis of the project social network (SNA). The first step in this analysis was to compute the social networks for each project. We counted each follow-up message in the bug tracking system as an interaction from the sender of the message to the preceding sender (or to the original bug reporter). Bug reports that had no follow-up messages provided no interaction data. It appeared from reading a sample of bug reports that follow-up messages were sometimes directed at previous messages and sometimes to the original poster. Unfortunately, the true destination is difficult to determine mechanically. We chose to code interactions as responses to the previous sender to spread out the interactions rather than focusing them on the bug poster. The arrows in Figure 2 show how two interactions were coded for this fragment of a bug report and messages. Note that follow-up messages are displayed in reverse chronological order in the system, so the response to the original report is actually the bottom message (not shown), the top message responds to the next, etc.

The interaction data from the bug reports form a network or graph, and were represented as a sociomatrix [15, p. 80], one matrix per project. A sociomatrix has a row and a column for each team member, and the cells of the matrix count the number of interactions from one member to another. If the interactions are directional, the resulting sociomatrices are asymmetric; if individuals can interact with themselves, the diagonals of the matrix are meaningful. Both conditions applied to our data.

The core and periphery analysis finds a partitioning of the group into core and peripheral members that most closely resembles the idealized core-and-periphery pattern described above. We carried out the analysis using the UCInet software package, which uses genetic algorithm to find the core/periphery split. The maximum number of iterations and population size were set to 5000 and 1000. The algorithm used in UCInet is equation 2 and 4 from Borgatti and Everett [1].

4. Results

We ran the analyses for the 116 SourceForge projects in our sample. Our first research question was to compare the results of the three different analysis approaches. As an example of the results, we will present in some detail the results for the Gaim project, one of the largest projects in the sample in the number of participants (1521). For this project, the identification of core group by the various methods, as shown in Table 2, shows some interesting differences. Gaim had 11 registered developers at the time of data collection, while the SNA core-and-periphery

Table 2. Counts of core/peripheral members for three methods for the Gaim project.

Developer status		SNA groups		Bradford's law groups	
Developer	11	Core	3	1st	2
Non-developer	1510	Periphery	1518	2nd	42
				3rd	1477

analysis found only 3 core members and the Bradford's law analysis, two developers as the core (these two together contributed more than 1/3 of the total number of messages) and a further 42 in the next group. Table 3 presents a cross-tabulation of these categorizations to show the level of agreement among them. To represent the 3-dimensions of the 2x2x3 comparison, the table shows developer status down, Bradford law groups across and SNA groups within each row. Table 3 shows that only one member is identified as in the core in all three analyses (the upper left-most cell). On the other hand, 1473 of the participants were grouped in the periphery by all three analyses (the lower right-most cell).

We used irr library from the R project statistical system (http://www.r-project.org/) to analyze the level of agreement between these classifications. Various statistics have been proposed to assess agreement of categorical classifications, with a score of 1.0 indicating perfect agreement and –1.0 perfect disagreement. One such statistic is the Finn coefficient; the Finn coefficient for the level of agreement between the three ratings (grouping together Bradford's law groups 2 and 3 as peripheral) was 0.989 (the overall level of agreement across all projects was 0.898). The high levels of agreement reflect the large number of peripheral members that are classified similarly by all three approaches. Another commonly used statistic is Cohen's kappa, which is simply the percent agreement corrected to take account the level of agreement expected by chance (so a zero score means just chance agreement). The classic kappa is defined for pairs of comparisons. Using this statistic, the measured level of agreement is lower: 0.140 between the developer list and the SNA analysis, 0.306 between the developer list and the Bradford law groups, and 0.399 between the SNA analysis and the Bradford's law groups, for an average kappa of 0.282. The levels of agreement between these ratings across all projects were 0.220, 0.208 and 0.464 respectively, for an average of 0.297. The kappa for the agreement between the developer list and the Bradford's law groups was higher when the Bradford's law groups 1 and 2 were considered as the core: 0.319 for gaim and 0.394 across all projects. These scores are lower than would be acceptable for qualitative coding (the usual target is 0.8) but all significantly greater than expected by chance.

To understand the differences between the classifications, we examined which individuals were identified as being in the core by each approach. Table 4 shows the core individuals as well as the next four highest contributors, including both project administrators. Closer examination of these results gives us some insight into the working of these different analysis approaches.

- `lschiere` was identified as a Group 1 member by the Bradford law analysis due to the high number of postings, but as a peripheral member in Core/Periphery. His role as Support Manager requires that he communicate frequently and with many different users, but in the data set we analyzed, he rarely communicated to the members of the SNA core (`warmenhoven`, `weeve` and `travissaling`), resulting in the peripheral classification.
- `warmenhoven` was identified as a Group 1 member by both the SNA and Bradford's law analysis. He was clearly an active developer in the project, posted a lot of messages and was identified as the center of core group by all of the methods.
- `seanegan`, `hermanator`, `thekingant` and `robflynn` were identified as Group 2 (peripheral) by both the SNA and Bradford's law analyses, despite their formal roles in the project. If we consider as the core Bradford's groups 1 and 2, then their formal and communication roles are aligned, but then the core

Table 3. Comparison of counts of core/periphery categorizations for three methods for the Gaim project.

	SNA groups	Bradford's law groups 1	2	3	Subtotal
Developers	**Core**	1	0	0	**11**
	Periphery	1	7	2	
Non-developers	**Core**	0	0	2	**1510**
	Periphery	0	35	1473	
Subtotal		**2**	**42**	**1477**	**1521**

Table 4. Comparison of classification of core individuals.

ID	Messages posted	Developer	Role	SNA group	Bradford's law group
lschiere	935	Yes	Support Manager	Periph	1
warmenhoven	839	Yes	Developer	Core	1
seanegan	444	Yes (Admin)	Developer	Periph	2
hermanator	259	Yes	Developer	Periph	2
thekingant	171	Yes	Developer	Periph	2
robflynn	112	Yes (Admin)	Project Manager	Periph	2
weeve	5	No	(User)	Core	3
travissaling	1	No	(User)	Core	3

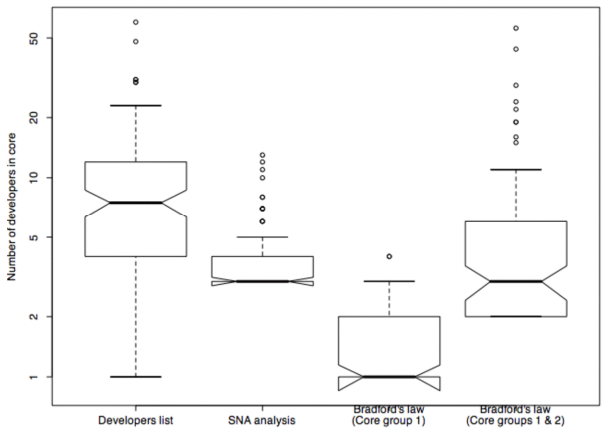
would include a total of 44 individuals, most of whom have no formal role. These results may indicate that these developers focused more on new development than on responding to bug reports, as these contributions are not reflected in our data set.

- The most surprising result is the identification of weeve and travissaling as core developers by the SNA analysis. Both are users who posted only a few messages, so they were in Group 3 in the Bradford's law analysis. However, they had communicated with warmenhoven regarding some bugs, making their connections to him dense, hence their inclusion with him in the core.

Given these results, it seems clear that the formal list of developers is not an accurate representation of contribution to the teams, at least as regards interactions around bug fixing. If developers specialize at all, then the list would likely be misleading for other parts of the process as well. Therefore, future research should make an independent attempt to empirically assess the size of the core group. The Bradford's law analysis is quite simple to perform and so may be the most useful for this purpose.

Our second research question was about the size of the core groups. Figure 3 compares the distribution of core groups sizes as computed by the different analysis techniques. The final two plots are for the core defined as Bradford's law group 1 only and by Bradford's law groups 1 and 2. The number of listed developers is lower than the cut-off of 7 for some projects because not all listed developers contributed to the bug tracking discussions. Nevertheless, this approach generally produced the largest count for the core (median 7.5, or about 10% of the total number of participants on average). The Bradford's law analysis using only group 1 as the core most often found only one core member (about 1.5% of the total on average) and never more than 4. This result demonstrates the extreme inequality in the level of contributions. (Our results differ from Mockus et al.'s [12] finding that bug reports are less centralized because they examined only the authorship of the initial bug report and not the follow-up messages.) The results for the SNA core-and-periphery analysis and the Bradford's law analysis (using groups 1 and 2 as the core) had comparable and intermediate results with a median of 3 developers in the core (about 5% of the total), though the results for the Bradford's law analysis were more extreme.

Figure 3. Box plots showing distribution of core group sizes as calculated by the three different analysis methods. Log scale. N=116.



Interestingly, these final numbers are quite compatible with James's [9] observations of the size of free-forming groups, suggesting that CMC does not alter the fundamental communications limits that are the bases for the group size. All three analysis techniques provide further support for the impression that FLOSS teams are highly centralized.

5. Conclusions

Consideration of the difference among the three analysis techniques suggests several possibilities for further refinement of the analysis. First, rather than using Bradford's level of 1/3 as the cutoff for the core group, it may be that a different level will provide a more useful definition of the core group, particularly given the high level of concentration observed.

Second, to avoid apparent anomalies such as weeve's and travissaling's membership in the core group,

the interaction matrix could be dichotomized with a cut-off greater than 1, thus admitting individuals to the core only if their level of interactions is both dense and higher than the cut-off.

Third, the bug-tracker data does not include contributions from developers who work on other aspects of the project, painting an incomplete picture of the project. Therefore, further analyses should include data from other kinds of interactions. Although there are significant problems with using such data (e.g., comparability among projects with different development practices, difficulties in identifying the target of interactions for an email message sent to a list, questions about how to weight interactions in different fora), using such data would pick up more of the intensive interactions among developers that characterize the core.

Fourth, the analysis should be extended to projects other than those on SourceForge. Though SourceForge provides a very convenient sample with extensive and comparable data on projects, our findings would be more clearly generalizable if we had data from other projects. At least, the sample should be redrawn to include more recently started projects.

Fifth, it may be worth considering how to develop a continuous measure of "coreness", rather than relying on a dichotomous definition. For example, to test the structure shown in Figure 1 requires at least 3 levels (core, co-developer and active user).

Finally, the analysis in this paper needs to be connected to work on other aspects of FLOSS teams to provide a fuller picture of the teams and to assess the importance of team structure for team performance. For example, a comparison of more and less effective FLOSS projects would reveal the relationship of structure to team performance. Analysis of particular development practices could compare core to peripheral members or focus in particular on the interactions of core team members.

6. References

[1] S. Borgatti and M. Everett, "Models of core/periphery structures," *Social Networks*, vol. 21, pp. 375–395, 1999.

[2] S. C. Bradford, *Documentation*. Washington, DC: Public Affairs Press, 1950.

[3] B. C. Brookes, "Theory of the Bradford Law," *Journal of Documentation*, vol. 33, pp. 189–209, 1977.

[4] A. Cox, "Cathedrals, Bazaars and the Town Council," 1998.

[5] K. Crowston, "A coordination theory approach to organizational process design," *Organization Science*, vol. 8, pp. 157–175, 1997.

[6] K. Crowston and J. Howison, "Hierarchy and Centralization in Free and Open Source Software team communications," *Knowledge, Technology & Policy*, In press.

[7] C. Gacek and B. Arief, "The many meanings of Open Source," *IEEE Software*, vol. 21, pp. 34–40, 2004.

[8] J. Howison, M. S. Conklin, and K. Crowston, "OSSmole: A collaborative repository for FLOSS research data and analyses," presented at 1st International Conference on Open Source Software, Genova, Italy, 2005.

[9] J. James, "A preliminary study of the size determinant in small group interaction," *American Sociological Review*, vol. 16, pp. 474–477, 1952.

[10] S. Krishnamurthy, "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," University of Washington, Bothell, Bothell, WA May 2002.

[11] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "A case study of Open Source Software development: The Apache server," in *Proceedings of ICSE'2000*, 2000, pp. 11 pages.

[12] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two Case Studies Of Open Source Software Development: Apache And Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 309–346, 2002.

[13] J. Y. Moon and L. Sproull, "Essence of distributed work: The case of Linux kernel," *First Monday*, vol. 5, 2000.

[14] E. S. Raymond, "The cathedral and the bazaar," *First Monday*, vol. 3, 1998.

[15] S. Wasserman and K. Frost, *Social Network Analysis: Methods and Applications*. New York: Cambridge, 1994.