# Hierarchy and centralization in Free and Open Source Software team communications[1]

Kevin Crowston and James Howison

School of Information Studies
Syracuse University

4-206 Centre for Science and Technology
Syracuse, NY 13244–4100

+1 (315) 443–1676
Fax: +1 (866) 265-7407
Email: {crowston, jhowison}@syr.edu

Please do not cite or quote

[1] This research was partially supported by NSF Grants 03-41475 and 04–14468. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# Hierarchy and Centralization in Free and Open Source software team communications

## Abstract

Free/Libre Open Source Software (FLOSS) development teams provide an interesting and convenient setting for studying distributed work. We begin by answering perhaps the most basic question: what is the social structure of these teams? Based on a social network analysis of interactions represented in 62,110 bug reports from 122 large and active projects, we find that some OSS teams are highly centralized, but contrary to expectation, others are not. Projects are mostly quite hierarchical on four measures of hierarchy, consistent with past research but contrary to the popular image of these projects. Furthermore, we find that the level of centralization is negatively correlated with project size, suggesting that larger projects become more modular. The paper makes a further methodological contribution by identifying appropriate analysis approaches for interaction data. We conclude by sketching directions for future research.

# Hierarchy and Centralization in Free and Open Source software team communications

Free/Libré Open Source Software (FLOSS) is a broad term used to embrace software developed and released under an "open source" license allowing inspection, modification and redistribution of the software's source without charge ("free as in beer"). Much though not all of this software is also "free software," meaning that derivative works must be made available under the same unrestrictive license terms ("free as in speech", thus "libré")[2]. In this paper, we investigate the informal social structure of FLOSS development teams by examining the pattern of communications between developers. Specifically, we measure the centralization and hierarchy of the communication in a number of projects. We are interested in team structure because, as explained in detail below, the practitioner-advocate literature on FLOSS often claims that particular aspects of social structure reflect and facilitate the advantages of FLOSS production. FLOSS advocates often link claims about its success and effectiveness to the distinctive and superior organization of their projects and communities. From Raymond's "Cathedral and Bazaar" (1998a) to Cox's "town council" and "cliques" (1998) to Krishnamurty's "Cave" (2002), organizational metaphors abound to characterize the organization of FLOSS projects.

Yet the little that we know about the organization of FLOSS comes largely from personal anecdote and case studies; empirical treatments are strongly needed. We are particularly interested in centralization and hierarchy within the teams because the predictions of the literature on these dimensions are somewhat mixed. Some strengths of the FLOSS approach are said to stem from organizational features that imply low centralization and hierarchy, while other appears to imply the opposite. To resolve this apparent discrepancy, we present measures of the structures of informal organization of FLOSS development projects using data drawn from three major FLOSS repositories: Sourceforge, the Apache Foundation and GNU's Savannah. These data answer the question: what is the typical structure, if any, of a FLOSS development team?

While FLOSS is an important research area in its own right, FLOSS teams are of more general interest because they are surprisingly successful examples of distributed teams. Distributed teams have been successfully applied to manage some very complex, large, and non-

[2] We have chosen to use the acronym FLOSS rather than the more common OSS to emphasize this dual meaning.

routine activities (Cutosksy *et al.*, 1996; Moon & Sproull, 2000). They seem particularly attractive for software development, because software, as an information product, can be easily transferred via the same systems used to support the teams (Nejmeh, 1994; Scacchi, 1991). Unfortunately, the problems of software development (e.g., interdependency and development of shared understandings) are exacerbated when development teams work in a distributed environment (Armstrong & Cole, 2002; Carmel & Agarwal, 2001; Curtis *et al.*, 1988; Grinter *et al.*, 1999; Herbsleb & Grinter, 1999).

While the literature on software development and distributed teams emphasize the difficulties of distributed software development, the case of FLOSS development presents an intriguing counter-example. Effective OSS development teams somehow profit from the advantages and evade the challenges of distributed software development (Alho & Sulonen, 1998). The "miracle of OSS development" poses a real puzzle and a rich setting for researchers interested in the work practices of distributed teams.

## Social structure, centralization and hierarchy

In this section, we briefly review the literature on FLOSS development that addresses the question of centralization and hierarchy. Centralization of a software project is usually thought of in terms of who writes the code (which we call development centralization). A highly centralized project by this definition is one in which a few of the members of the project team write most of the code, while a decentralized project would have a more equal division of labour.

A hierarchical structure is one that is organized or classified according to rank or authority. The stereotypical hierarchy is an organization with tiers of employees from the bottom to the top, linked by reporting relationships. However, the concept can be applied to connections other than reporting. A project with sharp divisions between a few developers and others would be hierarchical in terms of code development, because the developers would form a tier that has more authority over the code than the others.

While the above discussion has presented centralization and hierarchy in terms of the code, FLOSS projects depend on an open development process that people can easily join and contribute, collectively shaping the direction of the project. Accordingly, in this paper we focus our attention on the overall structure of communications in the project, rather than on programming in particular. Centralization of communications can be determined in a variety of ways, as discussed below, but the basic notion is that in a centralized structure a few individuals take part in most of the communications, while a decentralized structure would have more equal levels of participation. Hierarchy also has several measures, but all involve the notion that the communications structure is organized in tiers, with individuals at one level communicating to those at the next level.

## Practitioner literature

Much of the practitioner literature on FLOSS development emphasizes the importance of a decentralized and non-hierarchical pattern of communications. Eric Raymond praises the openness of FLOSS projects to suggestions and involvement from outside the initial developer or core team, "the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved." (Raymond, 1998a). Kuwabara (2000) characterises Raymond (1998a) as suggesting that the Linux model is "decentralized development" surrounded in "clamor ... anyone is welcome—the more people, the louder the clamor, the better it is". Similarly, in interviews we have conducted, developers associated with the Apache Foundation express a belief that non-hierarchical and decentralized structures are preferred because they are more robust to personality disputes and the withdrawal of individuals at the centre or top of the hierarchy. This structure is placed in marked contrast to the top-down 'cathedral' of proprietary software engineering centered on the "architect."

Linus' Law, introduced by Raymond (1998a), states a key purported strength of FLOSS development: "Given enough eyeballs, all bugs are shallow." This law is often relied upon when arguing that FLOSS development leads to more secure and less buggy products. Raymond reports that Linus Torvalds, the founder of Linux, notes "the person who understands and fixes the problem is not necessarily or even usually the person who first characterizes it." Instead, "Somebody finds the problem ... and somebody else understands it." The effectiveness of Linus' Law seems to rely on the absolute numbers of participants, developers, co-developers and active users in a project. Nonetheless it also appears to require that the communications structure of bug-fixing will include many participants, each expanding on reports, providing alternative conceptualizations of the problem or attempting solutions. Such a communications structure would tend to be decentralized and low in hierarchy, as anyone can communicate with anyone else.

Alan Cox describes an unintentional field experiment of these factors in his analysis of the Linux on 8086 project (Cox, 1998). In that project, the developers (including Cox himself) "walled themselves off" from other participants by using a 'kill-file' on their email. This technique allowed the central developers to not read all the messages on the mailing list—those from persons in the 'kill-file' go straight to the trash, unread and not responded to. The resulting communications structure would be quite hierarchical, as the core developers would form one tier, talking only to each other, while other participants would be in a different tier, talking to but not getting responses from developers. This 'kill-file' attitude was, Cox argues, an understandable but ultimately wrong response. In fact, Cox describes his experiences as "a guide to how to completely screw-up a free software development project". Cox argues that no matter how important it is to reduce distractions from 'town councilors,' excluding them from discussions is too harmful to the practice of free software to be considered. Instead the project must stimulate and permit broad discussion (keeping discussions focused on code that exists rather than merely ideas) and not let cliques place themselves above ordinary participants in the projects. As Raymond (Raymond, 1998a) puts it, there is a norm for even the founders of projects to "speak softly". In short, many of the leading FLOSS practitioners argue from their own experiences for the importance of a decentralized and non-hierarchical team structure.

While the literature reviewed above suggests the importance of non-centralized and non-hierarchial communications, the practitioner-advocate literature also praises social-structures that appear to be more hierarchical. For example Raymond (1998b) spends significant time discussing the concept of 'ownership,' which he identifies as an exclusive right to redistribute modified versions. He suggests that this right of 'ownership' is informal but strongly normative and often persists in the same individual that founded the project—a feature found, for example, in the Linux project. While Raymond clearly endorses "speaking softly", strong ownership ought to lead to patterns of deferral and authority that would reveal themselves in hierarchical communication patterns, with the owners on top. These patterns might also emerge because some developers become more frequent contributors and thus gain reputation and status within the group.

**Academic literature**

The academic literature examining FLOSS development is rapidly growing and has begun to investigate the social structure of projects. Researchers have studied both size and action patterns, concentrating on code production, but few have addressed interaction between project members. Primary among the currently published FLOSS research have been a number of case studies (Cox, 1998; Gacek & Arief, 2004; Mockus *et al.*, 2000, 2002; Moon & Sproull, 2000). That there have been only a limited number of many-project studies reflects the early stage of this research on this subject, the complexity of the phenomenon and the difficulty in obtaining comparable data across projects.

Krishnamurthy (2002), one of the limited number of many-project studies, challenged the belief that FLOSS projects are typically team-based at all. While his study was limited to the top 100 projects on SourceForge, he found a surprising number of one developer projects and a very strong skew to small developer teams, which was confirmed in our preliminary data. It is possible that this skew reflects the large number of still-born or 'code-dumped' projects that are hosted on SourceForge. Since one person contributes all of the code, one developer projects are centralized and hierarchical in development. However, it is an open question as to whether they are also centralized in their overall communications structure when including communications from users. (Of course, it is also an open question as to whether scholarship on FLOSS practices should take a great interest in single person projects or read a great deal about the effectiveness of FLOSS practices into their performance.)

Together the academic case studies of FLOSS projects {e.g., \Cox, 1998 #2399; Gacek, 2004 #2692; Moon, 2000 #2411; Mockus, 2000 #2720; Mockus, 2002 #2719} suggest a hypothesized model of FLOSS development with a hierarchical structure. The focus of these studies has largely been on the contribution of code. For example, Mockus et al. (2002) studied the Apache `httpd` project and found rapidly decreasing centralization from new code contribution, to bug-fixes to bug reporting. They 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).

This hierarchical or onion-like structure is shown in Figure **Error! Reference source not found.**. 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 forums.

Insert Figure 1 about here.

While these studies suggest that only a few developers are heavily involved in code production, they only briefly touch on the question of communications structure. Moon and Sproull (2000), in their case study of the development of Linux, describe the highly skewed distribution of traffic on the Linux mailing lists but their analysis is limited to counting postings and they do not examine interactions. That is, they describe who is talking, but not who is talking to whom. Similarly Mockus et al. (2002) describe patterns in initial bug-reports but do not examine patterns of communication which follow up these reports and which show the FLOSS development process in action.

However, there is reason to believe that the model in Figure 1 extends as well to communications. In discussions with developers, we have identified the development of a buffer of active users as a desirable FLOSS community feature for communications reasons. Active users create a rich support and their collected answers form a knowledge base that others can draw on to answer questions. The availability of willing and able user-to-user support is an often cited benefit of using FLOSS and is said to be of higher quality than commercial support. This clearly describes a hierarchical structure with the developers in the centre, active users forming a "middle management" layer, and the users seeking support at the periphery.

As well, much of the research on motivations of FLOSS developers points to the importance of status and prestige (Ghosh, 2002; Hertel *et al.*, n.d.; Ye & Kishida, 2003). This is the foundation of the purported gift economy {Bergquist, 2001 #2757} that has entranced economists looking at FLOSS, as well as something participants themselves point to. If FLOSS is an activity which leads to the emergence of status and prestige we would expect communications to reflect this hierarchy (Krackhart, 1993). We therefore decided to test for the emergence of hierarchy in order to empirically test the hypothesis that FLOSS is a status forming activity.

It is these interactions that are the focus of our study. In summary then, the literature, while no means all in agreement, leans towards a prediction of hierarchy in FLOSS project communication structure.

## Methods and data

To explore the structure of FLOSS teams empirically, we employed social network analysis (SNA). The first step in the analysis was to collect interaction data for a sample of projects. The analysis reported in this paper is of interactions related to the bug fixing process for FLOSS projects. We chose to study bug fixing because it provides a "microcosm of coordination problems" (Crowston, 1997) and is a collaborative task in which, as Raymond (1998a) paraphrases Linus Torvalds, the people finding the bugs are different from those that understand the bug and those that fix the bug. Thus the collaboration of bug fixing produces rich interactive collaborations and it is a process that involves 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. Furthermore, the use of bug tracking systems provide a convenient sample of bug-related discussions and one in which interactions could be defined. Finally, as described above, bug fixing is the site of some claims of effectiveness made for FLOSS projects.

### Data source

To study the network of interactions around bug fixing, we collected interaction data from bug tracking systems. In addition to tracking the status of bugs, these systems enable users to report, and developers to discuss bugs. 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 social structure of the teams. We collected data from three bug tracking systems: the SourceForge tracker, the Apache Foundation Bugzilla and the Savannah bug tracker. The diversity of projects is a strength of our study, since most analyses are restricted to a single system.

Insert Figure 2 about here.

The first sample of projects was drawn from projects hosted by SourceForge (http://www.sourceforge.net/), a free[3] Web-based system that provides a range of tools to facilitate OSS development. At the time of our data collection, SourceForge supported more than 50,000 OSS projects on a wide diversity of topics[4]. 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 (Krishnamurthy, 2002), 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 (and were able to obtain data for 122, as discussed below). Our second sample was drawn from the GNU Savannah system (http://savannah.gnu.org), which support 2,208 projects, 281 of which are part of the GNU project. The bug tracking system lists 372 projects, but only 22 of these had more than 100 bugs, so these 22 were used for the analysis. Our third and final sample was drawn from the 56 projects using the Apache Software Foundation Bugzilla bug tracking system. Thirty-two (32) of these projects had more than 100 bugs in the system and were used.

Unfortunately, space does not permit a full listing of the projects, but Table 1 lists examples to give a sense of the samples. Those familiar with OSS may recognize some of these projects, which span a wide range of topics and programming languages.

Insert Table 1 about here.

**Coding**

Two key issues in the application of SNA are the definition of an actor and of an interaction. 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 actors. It is possible (and

[3] At least free 'as in beer': ironically, the SourceForge system itself is now proprietary. Savannah was developed from the last free 'as in freedom' version of SourceForge.

[4] As of 12 December 2004, SourceForge claims 92,181 projects.

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.

The definition of an interaction was more involved. For this analysis, we counted each follow up message in the bug tracking system as one 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 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 (Wasserman & Frost, 1994, p. 80), one matrix per project. A sociomatrix has a row and a column for each individual, and the cells of the matrix count the number of interactions from one individual 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.

To collect data from SourceForge, we developed programs to download and parse the bug report pages for the selected projects. 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. Interactions for the Apache and Savannah projects were extracted directly from the databases, which were obtained for research purposes from the foundations that run them. The databases were provided in November 2004. The data we processed to obtain the sociomatrices are summarized in Table 2.

Insert Table 2 about here.

Processing of 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. We rejected the simplest solution of counting all nobody messages as being from the same sender (“nobody”) because the large number of such messages would have seriously biased our results. Simply dropping the “nobody” interactions from the sociomatrix by deleting the “nobody” row and column had the disadvantage of underestimating the number of participants in the discussion and leaving some participants disconnected from the discussion. In the end, we 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. The effect on the count of developers is not clear: it may increase the estimate because the same anonymous person might post in multiple bug reports, but be counted separately; or it may decrease the count because multiple anonymous users posting in a single bug report are counted as one.

## Analysis

Once we had the sociomatrices, we examined the projects’ structures. Following an exploratory data analysis philosophy, we first plotted the interaction graphs for a selection of projects to visualize the interactions and get a sense of the data. Figure 3 shows the interaction plot for a typical project, *openrpg*, created in the program Netminer and hand edited to reduce the number of labels. Note that in an interaction plot, the important information is the pattern of connections between nodes rather than their precise locations (that is, the X-Y dimensions of the graph are not interpretable). The distance between nodes is an approximation of the strength of the ties, e.g., using Kamada and Kawai’s (1989) spring embedding algorithm. The plot in Figure 3 suggests that the interactions in this project are centered on a few individuals, and that the peripheral individuals have typically only posted a bug report (indicated by having only an arrow coming in), consistent with the hypothesized structure.

Insert Figure 3 about here.

To numerically test the hypothesis that projects have centralized and hierarchical structures, we calculated the network centralization and four measures of hierarchical structure

(Krackhart, 1993). All five measures range from 0 to 1. Centralization has been long studied in the field of Social Network Analysis and has several accepted definitions {Wasserman, 1994 #2981}. For centralization, 0 means a perfectly decentralized network (e.g., a ring or totally connected graph) while 1 means perfectly centralized network (a star). Hierarchy is a less commonly studied concept. The intuitive meaning of the term is clear, but it is more difficult to reduce to a single number. Instead, Krackhart suggests four different measures of a network's structure: connectedness, hierarchy, efficiency and lubness. These are defined to be 1 for a true hierarchy, and 0 for a non-hierarchy. The details of each measure is given below. The calculations we report were computed using Jeff Reminga's NetStatPlus library (http://www.casos.cs.cmu.edu/projects/netstat/).

*Network centralization*

The calculation of network centralization starts by determining the centrality of each individual in the network. There are many different definitions of centrality in the literature (Wasserman & Frost, 1994, Ch. 5), and the choice between measures is based on the substantive nature of the interactions. We decided to use the most basic definition, which is based on degree: individuals who receive or send more connections are more central than those that do not. This choice was based on the interpretation that an individual who posts messages in reply to more bug reports is more central to the bug fixing process than one who posts fewer. Alternative definitions of centrality, such as those based on betweenness, proximity or closeness, seemed not to fit the data we had collected. The usual calculation of degree centrality is based on dichotomous data (i.e., communication vs. no communication). We dichotomized the sociomatrices with 1 message as the cut-point, so individuals' centralities were simply the count of the individuals with whom they interacted (possibly including themselves). The degree counted can be the in-degree (number of interactions received), out-degree (number sent) or sum of the two (sometimes called the Freeman centrality). It is typical to attribute centrality to an individual who receives a lot of messages (in-degree centrality), but we chose to compute out-degree centrality because of our interest in identifying individuals who contribute to a broad range of bug reports.

Once we had calculated the individual centrality measures, we could calculate the centralization of the entire network. The standard definition of network centralization is based on the inequality of the individual centrality measures: in a very centralized network, one individual will have a high centrality and the others low, while in a decentralized network, no single individual will stand out, i.e., all the centralities will be about the same, high or low (Wasserman & Frost, 1994, p. 176). Specifically, the network centralization is calculated as the sum of the differences between the maximum and each individual's centrality score, normalized to range from 0 to 1 by dividing by the theoretical maximum centralization. The theoretical maximum is the centralization that would be obtained in a perfectly centralized star network where the only interactions are a central individual talking to everyone else. Figure 4 presents three box plots showing the distribution of centralization scores for projects from the three data sources. Note the wide range of scores for this measure, indicating that projects vary greatly in how centralized they are.

Insert Figure 4 about here.

*Connectedness*

The remaining four measures measure different aspects of a network's structure, comparing specific properties of a given network to the properties that would hold for an ideal hierarchy (one organized into tiers with relations between the tiers). The first measure is connectedness. In an ideal hierarchy, every member of the organization is connected to someone else in the organization, and so the connectedness score is defined as 1 for a connected graph (one in which every node can reach every other node in the underlying undirected graph.) A fully unconnected graph has no edges at all, and so gets a connectedness score of 0. Connectedness seems to be useful in ways similar to centrality, as it reveals if there is one team or whether some members are not connected, possibly working on their own or in a smaller subgroup. Figure 5 shows the distribution of connectedness scores for projects.

Insert Figure 5 about here.

*Graph Hierarchy*

The second measure is graph hierarchy. Krackhardt (1993) notes that in an ideal hierarchical organization, relationships are asymmetrical. There are no loops in which, for example, one person's subordinate is also that person's boss's boss. Graph hierarchy is defined to range from 0 in a network in which all relationships are symmetrical to 1 in which they are all asymmetrical. Krackhardt (1993) suggests that this measure can be seen as operationalizing, "the degree to which the organization is dominated by status in its informal relations." This interpretation can be applied to our interaction data, as it would indicate that individuals who report bugs rarely respond to the comments of other, presumably more central developers who comment on the bugs. The distribution of hierarchy scores for projects is shown in Figure 6.

Insert Figure 6 about here.

*Efficiency*

The third measure is efficiency. In an ideal hierarchy, every individual has a single rather than multiple superiors, so a graph is considered graph efficient if there are not multiple paths between nodes. Krackhardt (1993) suggests that this measure will have a "curvilinear relationship to organizational effectiveness". In a perfectly efficient network, the loss of a single link will leave the network disconnected, meaning that the network is brittle and performance will be reduced for very high levels of efficiency. On the other hand, density in relationships (and thus lower efficiency) might be good for robustness of the organization but, as the efficiency reduces, the increased overhead of maintaining all these linkages would reduce performance. Efficiency is closely related to the density of the network; as density increases, efficiency decreases. In our data, a dense network would be formed if many individuals responded to messages from many others, rather than just a few. Figure 7 shows the distribution of efficiency scores for projects.

Insert Figure 7 about here.

*Least Upper Boundness*

The final measure is "lubness". In an ideal hierarchy, all pairs of individuals have a common superior somewhere higher up the hierarchy. Lubness (short for least upper boundedness), measures the extent to which pairs of individuals in the network communicate directly or indirectly with a common individual. It is defined to be 1 for a perfect hierarchy in which all pairs have a common superior and 0 where none do. Krackhart (1993) suggests that lubness is associated with ease of resolving "organizational conflict" because the higher lubness, the more there will be a person ultimately in charge and able to resolve the conflict. For our communications networks, lubness indicates the degree to which the network has a single centre:

an individual who (eventually) answers everyone's question. Figure 8 shows the distribution of lubness scores for the projects.

Insert Figure 8 about here.

*Interactions*

To explore the potential of the centralization and hierarchy measures for understanding the structure of project groups, we calculated the correlation between the scores and other measures of interest. Specifically, we calculated Pearson correlations coefficients among the scores and with the number of developers who contributed messages to the bug report tracking system, which we used as a measure of the overall project size. The count of developers was heavily skewed, so it was log transformed for analysis. This transformation is justified theoretically, since the size of the project is the result of some kind of growth process. The correlation coefficients are shown in Table 3.

Insert Table 3 about here.

## Discussion

Our simple data analysis revealed several surprising findings. First, to our surprise, our data indicate that OSS projects are not uniformly centralized nor hierarchical, contrary to expectations. In fact, the calculated centralization measures display a considerable range, as shown in Figure 3. In short then, our data demonstrate that the centralization of OSS projects is in fact distributed, with a few highly centralized projects, a few decentralized and most somewhere in the middle (indeed, the mean centralization is 0.54).

Second, most projects have high connectedness scores, indicating that contributors are mostly linked into the communications structure. The lower scores for some SourceForge projects indicates that some contributors are not interacting with the rest of the group, perhaps because the bugs they have reported have not yet been addressed by core developers. Connectedness may be an interesting measure for further analysis. Calculating connectedness after raising the threshold for dichotomizing the interaction matrix (an analysis we did not perform) would reveal how the network breaks into clusters with high levels of communications and thus measure modularity in communication structure.

Third, the hierarchy scores for projects show some distribution, with a mean of 0.80. The relatively high level suggests that most participants in the discussions do not reciprocate messages. This pattern would hold if most bug reports are content to simply report the bug and not take part in the follow up discussions. This is an interesting finding because it contradicts the descriptions in the practitioner literature of FLOSS teams as "surrounded in 'clamor'". The high efficiency and lubness scores further the impression of essentially one-way communications in most projects. The high efficiency scores also suggest that many participants are only weakly connected to the organization.

Fourth, project centralization scores are strongly negatively correlated with number of participants in the bug report discussions. The plot of this relationship, shown in Figure 9, suggests that small projects can be centralized or decentralized, but larger projects are decentralized. We interpret this finding as a reflection of the fact that in a large project, it is simply not possible for a single individual to be involved in fixing every bug. As projects grow, they have to become more modular, with different people responsible for different modules. In other words, a large project is in fact an aggregate of smaller projects, resulting in what might be described as a "shallot-shaped" structure, with layers around multiple centres. Figure 10 shows the interaction graph for a highly centralized project, curl; Figure 11, the graph for a larger decentralized project, squirrelmail.

Insert Figures 9, 10 & 11 about here.

Finally, there is a high negative correlation between project size and density. In fact, the relationship is an inverse one, as shown in Figure 12. Graph density is the ratio of the actual links to the possible links in a completely connected graph, which increases as the square of the number of participants. The relationship shown in Figure 12 suggests that the number of connections a participant can maintain remains roughly constant as the project grows, making the network increasingly less dense.

Insert Figure 12 about here.

## Conclusion

Many authors have speculated on the structure of FLOSS teams. In this paper, we present data that demonstrates that at least for discussions around bug fixing, project teams social structures are largely hierarchical consistent with the academic literature we reviewed. On the other hand, the projects display a surprising range of centralizations, with some teams being highly centralized and others decentralized. However, there is a tendency for large projects to be less centralized.

Our study has the strength that it uses data from three sources, rather than just one. However, it does suffer from a limitation in that all of the projects selected are basically successful in that they have attracted developers and generated activity in the form of bug reports and discussion (Crowston *et al.*, 2003). A future study should compare the structure of these successful projects to other less successful projects to assess how the structure affects the performance of the teams. As well, it would be instructive to compare the communications patterns we have observed in FLOSS teams to those of proprietary software development teams.

As well, the study examined only interactions around bug fixing in the bug tracking system. From conversation with developers, we have learned that projects differ greatly in how they use these tools; some do not use them at all, and some that do use them do not use them for discussions, preferring email instead. A future study should incorporate data about interactions from multiple sources.

Of course organizational hierarchies are not static, but rather develop and evolve over time. A limitation of our analysis is that we collapsed all interactions across time into a single measure. A future study might analyze the interactions over time to provide a more dynamic picture of the evolution of communications networks. Since all of the messages in the bug trackers are time stamped, it is possible to do so. This type of analysis might reveal whether decentralized projects have multiple centres at the same time or whether one individuals acts at the centre for a while, and is then replaced by another person, giving the appearance of multiple centres for the network when analyzed as a whole.

Finally, our analysis has examined the structure of project communication networks but not their content. A next step for studies of FLOSS will be to examine what the individuals are actually saying and how these conversations support productive software development practices. Even so, information from these sorts of analyses will be helpful in identifying where in the process each participant is situated and in identifying differently shaped projects.

## Bibliography

Alho, K., & Sulonen, R. (1998). *Supporting virtual software projects on the web*. Paper presented at the Workshop on Coordinating Distributed Software Development Projects, 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98).

Armstrong, D. J., & Cole, P. (2002). Managing distance and differences in geographically distributed work groups. In P. Hinds & S. Kiesler (Eds.), *Distributed work* (pp. 167–186). Cambridge, MA: MIT Press.

Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software*(March/April), 22–29.

Cox, A. (1998). Cathedrals, bazaars and the town council. Retrieved 22 March 2004, from http://slashdot.org/features/98/10/13/1423253.shtml

Crowston, K. (1997). A coordination theory approach to organizational process design. *Organization Science, 8*(2), 157–175.

Crowston, K., Annabi, H., & Howison, J. (2003). Defining open source software project success. In *Proceedings of the 24th international conference on information systems (icis 2003)*. Seattle, WA.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *CACM, 31*(11), 1268–1287.

Cutosksy, M. R., Tenenbaum, J. M., & Glicksman, J. (1996). Madefast: Collaborative engineering over the internet. *Communications of the ACM, 39*(9), 78–87.

Gacek, C., & Arief, B. (2004). The many meanings of open source. *IEEE Software, 21*(1), 34–40.

Ghosh, R. A. (2002). Free/libre and open source software: Survey and study. Report of the floss workshop on advancing the research agenda on free / open source software. from http://www.infonomics.nl/FLOSS/report/workshopreport.htm

Grinter, R. E., Herbsleb, J. D., & Perry, D. E. (1999). The geography of coordination: Dealing with distance in r&d work. In *Proceedings of the group '99 conference* (pp. 306–315). Phoenix, Arizona, US.

Herbsleb, J. D., & Grinter, R. E. (1999). Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*(September/October), 63–70.

Hertel, G., Niedner, S., & Herrmann, S. (n.d.). Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel. Kiel, Germany: University of Kiel.

Kamada, T., & Kawai, S. (1989). An algorithm for drawing general unidirected graphs. *Information Processing Letters, 31*, 7–15.

Krackhart, D. (1993). Graph theoretical dimensions of informal organizations. In K. M. Carley & M. J. Prietula (Eds.), *Computational organization theory* (pp. 89–111). Hillsdale, NJ: Lawrence Erlbaum.

Krishnamurthy, S. (2002). Cave or community? An empirical examination of 100 mature open source projects. Bothell, WA: University of Washington, Bothell.

Kuwabara, K. (2000). Linux: A bazaar at the edge of chaos. *First Monday, 5*(3).

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2000). A case study of open source software development: The apache server. In *Proceedings of icse'2000* (pp. 11 pages).

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309–346.

Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of linux kernel. *First Monday, 5*(11).

Nejmeh, B. A. (1994). Internet: A strategic tool for the software enterprise. *Communications of the ACM, 37*(11), 23–27.

Raymond, E. S. (1998a). The cathedral and the bazaar. *First Monday, 3*(3).

Raymond, E. S. (1998b). Homesteading the noosphere.

Scacchi, W. (1991). The software infrastructure for a distributed software factory. *Software Engineering Journal, 6*(5), 355–369.

Wasserman, S., & Frost, K. (1994). *Social network analysis: Methods and applications*. New York: Cambridge.

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. In *Proceedings of 2003 international conference on software engineering (icse2003)*. Portland, OR.

## Figures and Tables

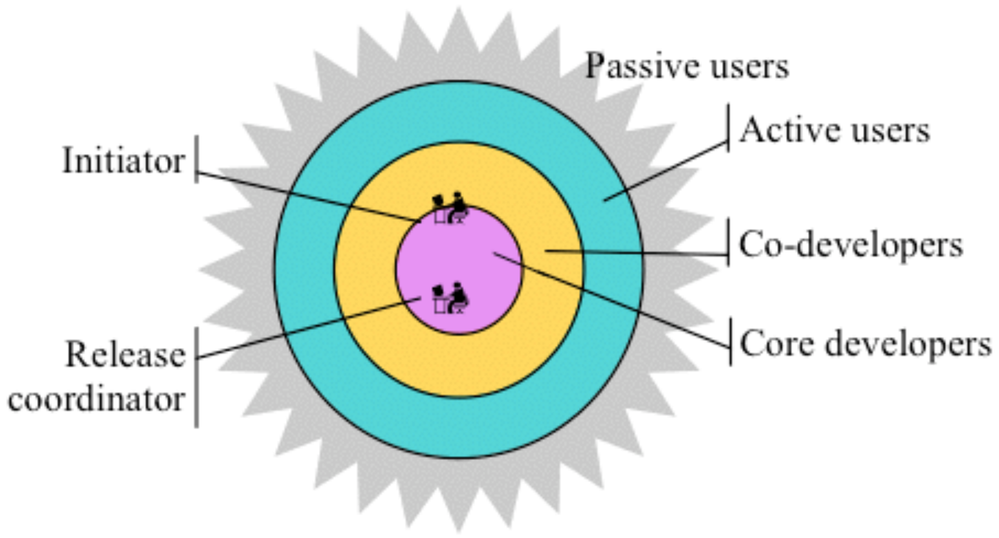
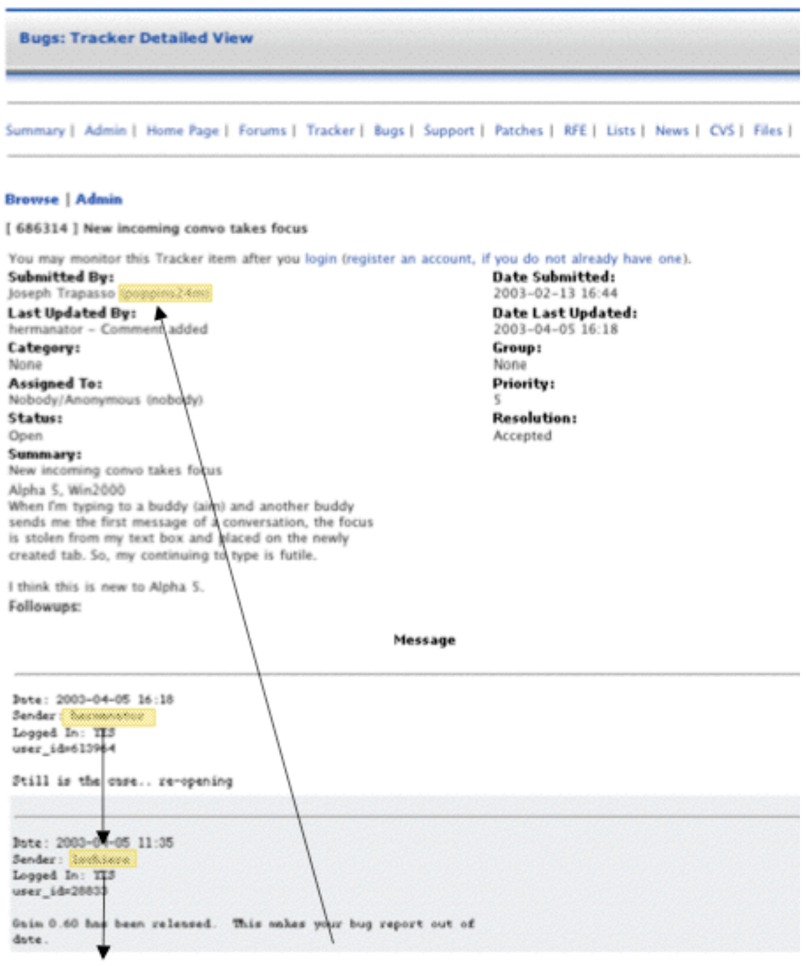


**Figure 1.** A synthesized FLOSS development team structure.


Bugs: Tracker Detailed View
Summary | Admin | Home Page | Forums | Tracker | Bugs | Support | Patches | RFE | Lists | News | CVS | Files |
Browse | Admin

[ 686314 ] New incoming convo takes focus

You may monitor this Tracker item after you login (register an account, if you do not already have one).

**Submitted By:**
Joseph Trapasso [illegible]

**Date Submitted:**
2003-02-13 16:44

**Last Updated By:**
hermanator - Comment added

**Date Last Updated:**
2003-04-05 16:18

**Category:**
None

**Group:**
None

**Assigned To:**
Nobody/Anonymous (nobody)

**Priority:**
5

**Status:**
Open

**Resolution:**
Accepted

**Summary:**
New incoming convo takes focus

Alpha 5, Win2000
When I'm typing to a buddy (aim) and another buddy sends me the first message of a conversation, the focus is stolen from my text box and placed on the newly created tab. So, my continuing to type is futile.

I think this is new to Alpha 5.

Followups:

**Message**

```
Date: 2003-04-05 16:18
Sender: hermanator
Logged In: YES
user_id=613964

Still is the case.. re-opening
```

```
Date: 2003-04-05 11:35
Sender: lschiere
Logged In: YES
user_id=28833

Gaim 0.60 has been released.  This makes your bug report out of
date.
```

**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

**Table 1.** Examples of projects included in sample.

| Project name | Source | Short description |
|---|---|---|
| curl | SourceForge | Command line tool and library for client-side URL transfers. |
| gaim | SourceForge | A GTK2-based instant messaging client. |
| netatalk | SourceForge | A kernel-level implementation of the AppleTalk Protocol Suite. |
| phpmyadmin | SourceForge | Handles the basic administration of MySQL over the WWW |
| squirrelmail | SourceForge | A PHP4 Web-based email reader. |
| tcl | SourceForge | Tool Command Language |
| GNU arch | Savannah | A revision control system |
| Make | Savannah | Compilation control system |
| DotGNU | Savannah | Portable .NET |
| httpd-1.3 | Apache | Webserver |

**Table 2.** Statistical summary of raw data processed to extract interaction data.

| | SourceForge | Savannah | Apache Bugzilla |
|---|---|---|---|
| **Number of projects** | 122 | 22 | 32 |
| **Bugs** | | | |
| Total number | 62,110 | 7,040 | 30,312 |
| Average /project | 509 | 320 | 947 |
| Median /project | 279 | 215 | 615 |
| **Interactions** | | | |
| Total number | | 10,675 | 84,101 |
| Average/bug | | 1.5 | 2.8 |
| **Participants** | | | |
| Total number | 14,922 | 1,348 | 16,579 |
| Average /project | | 65 | 606 |
| Number in multiple projects | 1,280 | 52 | 2,204 |

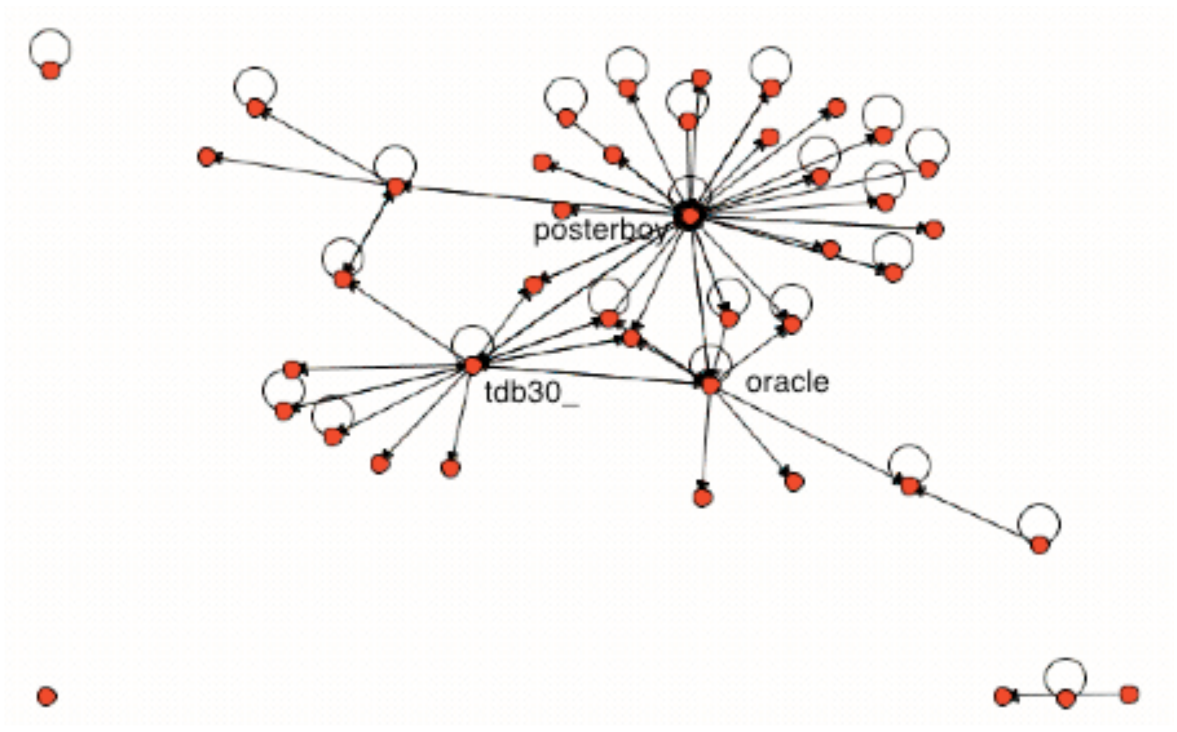


**Figure 3.** Plot of interactions for the openrpg project bug report data, created in Netminer.

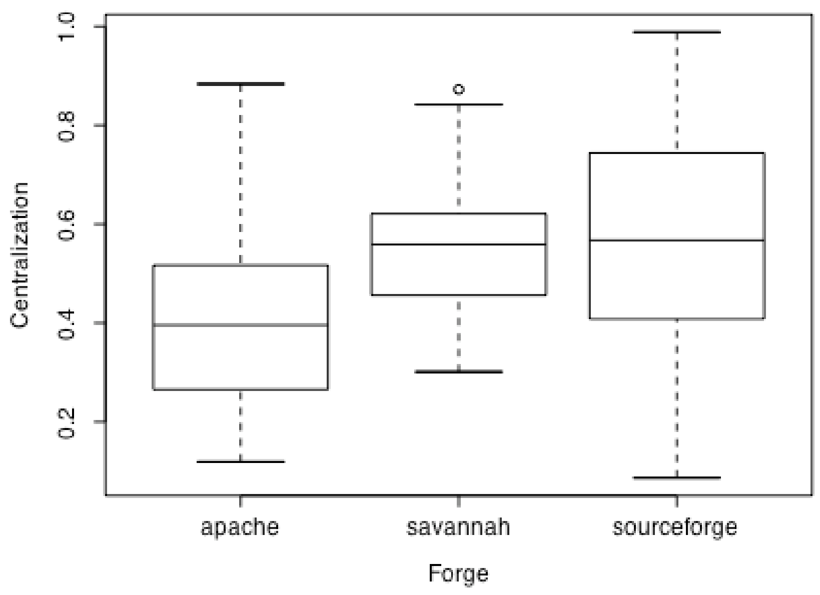


**Figure 4.** Distribution of out-degree centralization scores for projects.

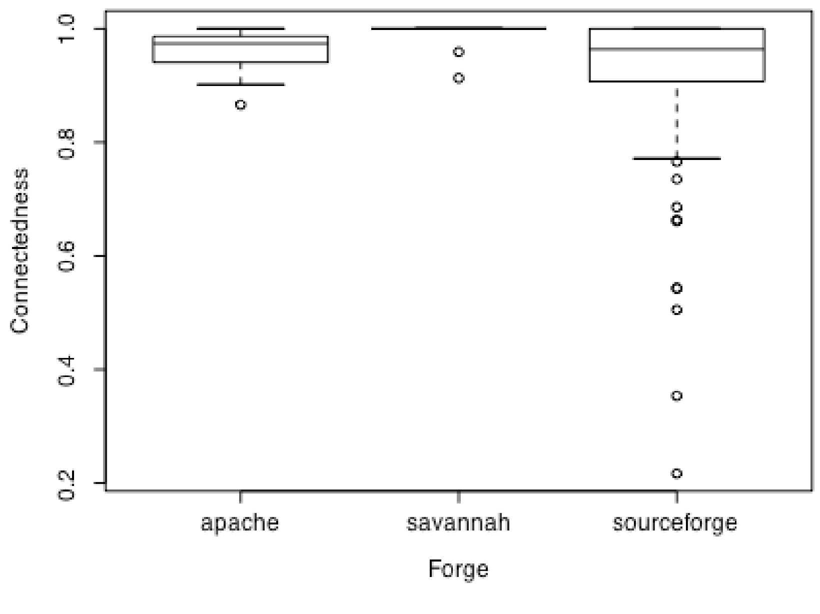


**Figure 5.** Distribution of connectedness scores for projects.

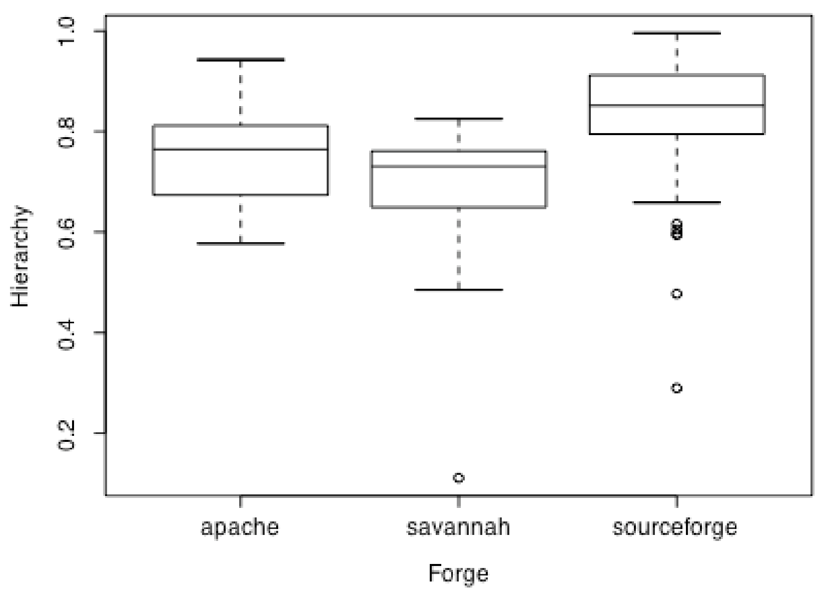


**Figure 6.** Distribution of graph hierarchy scores for projects.

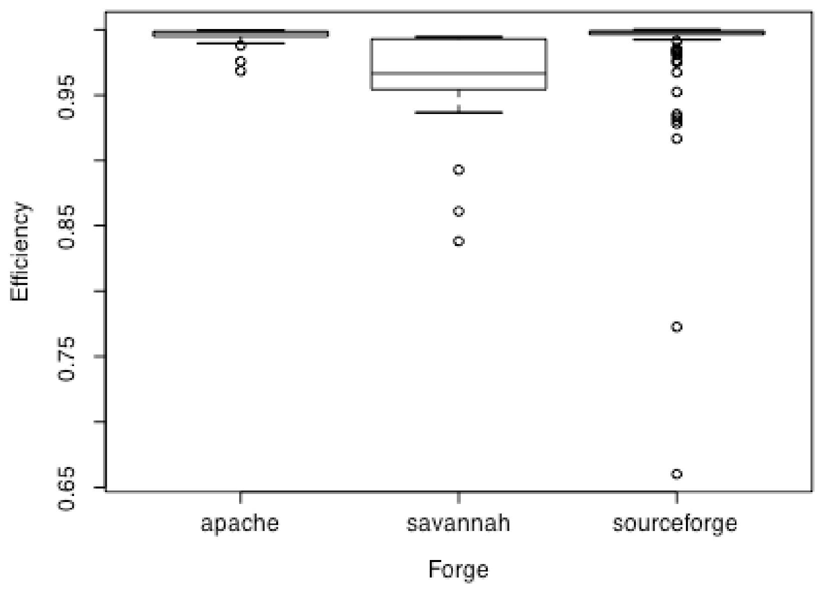


**Figure 7.** Distribution of efficiency scores for projects.

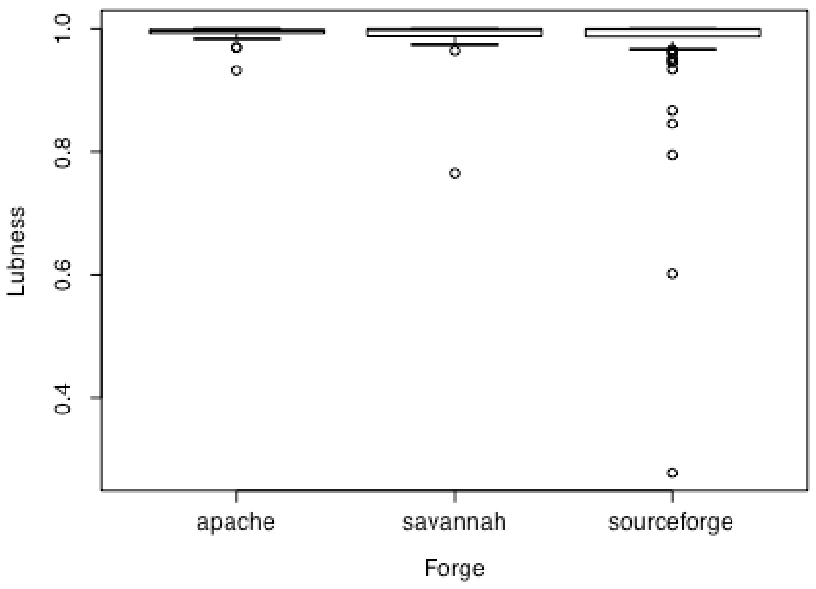


**Figure 8.** Distribution of lubness scores for projects.

**Table 3.** Correlation coefficients among network measures.

| | **log N** | **Deg** | **Den** | **Con** | **Hie** | **Eff** | **Lub** |
|---|---|---|---|---|---|---|---|
| **log.N.** | 1.00 | -0.40 | -0.71 | 0.06 | 0.10 | 0.51 | 0.03 |
| **Degree** | -0.40 | 1.00 | 0.15 | 0.29 | 0.09 | -0.03 | 0.18 |
| **Density** | -0.71 | 0.15 | 1.00 | 0.11 | -0.44 | -0.92 | 0.01 |
| **Connectedness** | 0.06 | 0.29 | 0.11 | 1.00 | -0.25 | -0.11 | 0.29 |
| **Hierarchy** | 0.10 | 0.09 | -0.44 | -0.25 | 1.00 | 0.53 | -0.19 |
| **Efficiency** | 0.51 | -0.03 | -0.92 | -0.11 | 0.53 | 1.00 | -0.02 |
| **Lubness** | 0.03 | 0.18 | 0.01 | 0.29 | -0.19 | -0.02 | 1.00 |

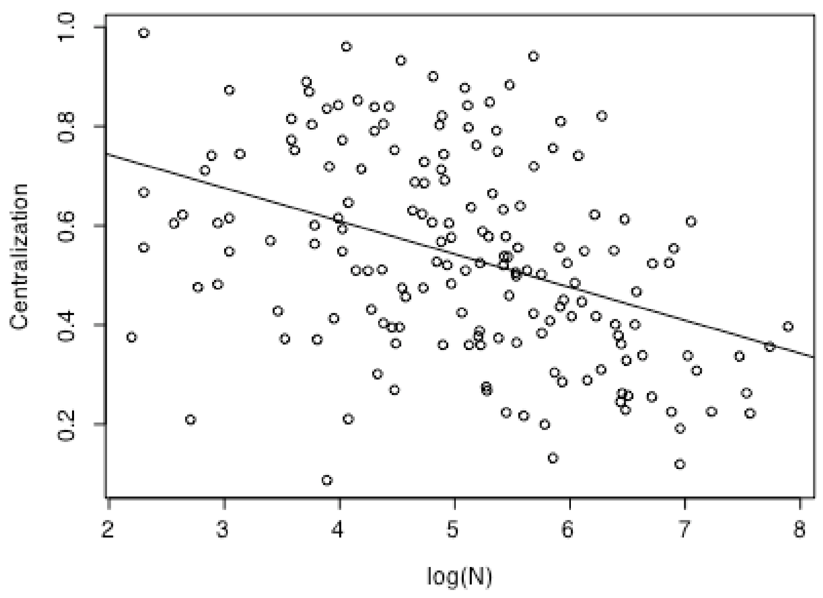


**Figure 9.** Centralization versus project size (r=-0.52).

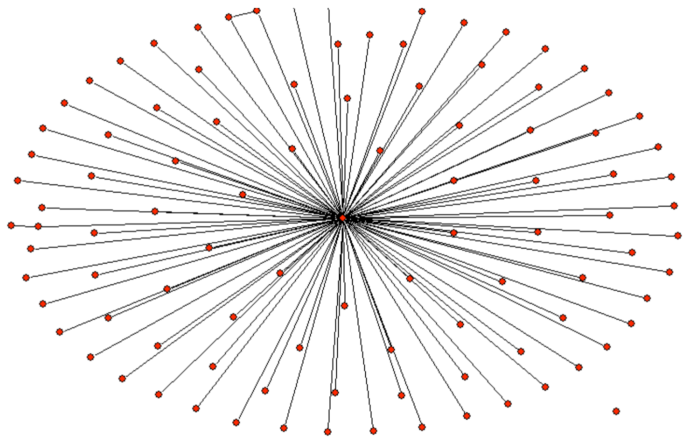


**Figure 10.** Plot of interactions for curl, a highly centralized project (centralization = 0.922).

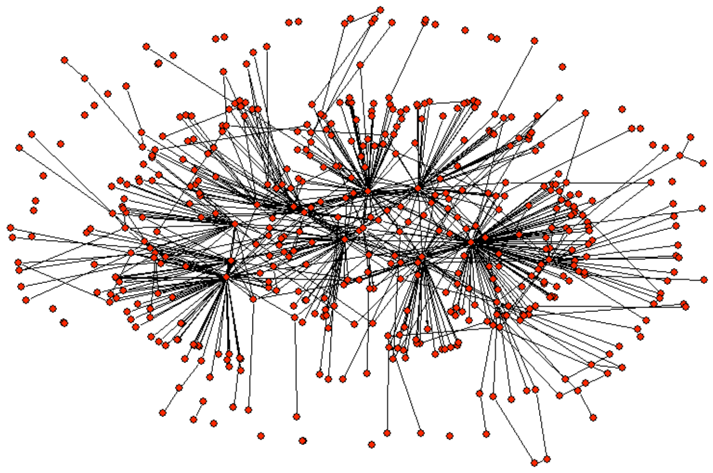


**Figure 11.** Plot of interactions for squirrelmail, a decentralized project (centralization = 0.377).

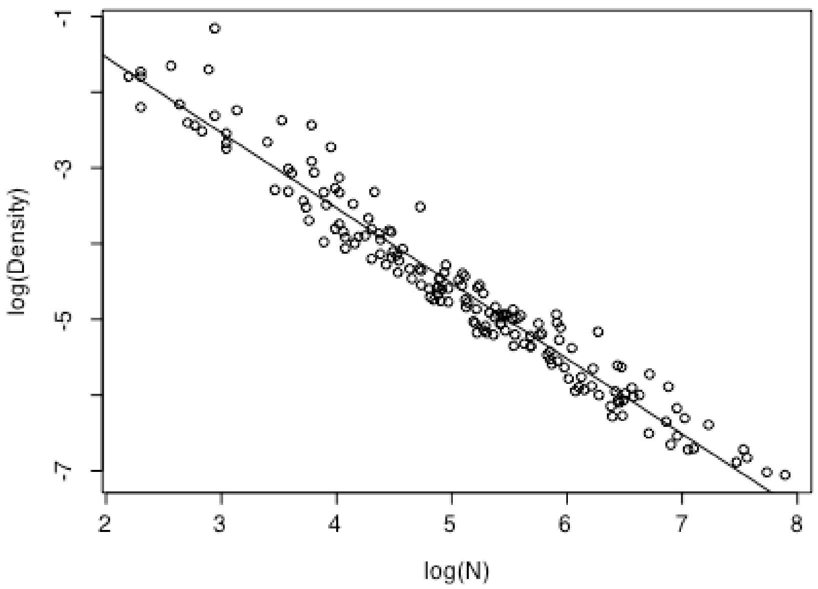


**Figure 12.** Density decreases as the inverse of network size.