The social structure of Open Source Software development teams

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@syr.edu, james@freelancepropaganda.com

Submission to the *Individuals, Teams, and Virtual Communities* Track
of the 2003 *International Conference on Information Systems*

Draft of 5 May 2003

The social structure of Open Source Software development teams

Abstract

Open Source Software 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. 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.

Completed Research Paper submission
to the *Individuals, Teams, and Virtual Communities Track*
of the *International Conference on Information Systems*

5813 words, 22 pages (1 title/abstract + 20 text)

Keywords: Open Source Software, software development, bug fixing, distributed work, social network analysis

The social structure of Open Source Software development teams

In recent years, organizations have become more reliant on distributed teams. In this paper, we report the initial step of a study of Open Source Software development teams as exemplars of distributed teams. Specifically, we present data on the social structure of large and active teams that shows that these teams exhibit a wide range of degrees of centralization, contrary to expectation, and that large teams tend to be less centralized.

Though distributed work has a long history (O'Leary, Orlikowski, & Yates, 2002), contemporary teams typically rely on computer-mediated communications (CMC) for communication and coordination, so advances in information and communication technologies have been crucial enablers for the development of this organizational form (Ahuja, Carley, & Galletta, 1997). As a result, distributed teams have become a lively area of research in the information systems field. Distributed teams offer numerous potential benefits, such as reducing costs associated with travel or relocation, ease of reconfiguring teams (DeSanctis & Jackson, 1994; Drucker, 1988) and exploitation of distributed expertise (Grinter, Herbsleb, & Perry, 1999; Orlikowski, 2002).

Distributed teams have been successfully applied to manage some very complex, large, and non-routine activities (Cutosksy, Tenenbaum, & Glicksman, 1996; Moon & Sproull, 2000). Distributed teams 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 are exacerbated when development teams work in a distributed environment with a reduced possibility for informal communication (Bélanger, 1998; Carmel & Agarwal, 2001; Herbsleb & Grinter, 1999a). The lack of a common organizational setting or functional background can make socialization, communication and coordination processes difficult, reducing team performance and increasing the need for coordination and learning among members (Armstrong & Cole, 2002; Finholt, Sproull, & Kiesler, 1990; Robey, Khoo, & Powers, 2000). Furthermore, because teams rely on CMC, it may be more difficult to develop the informal relationships and communications necessary to address these interpersonal issues (Curtis, Krasner, & Iscoe, 1988; Grinter et al., 1999).

While the literature on software development and distributed teams emphasize the difficulties of distributed software development, the case of Open Source Software (OSS) development presents an intriguing counter-example. OSS is a broad term used to embrace software that is developed and released under some sort of "open source" license (Gacek, Lawrie, & Arief, n.d.). There are thousands of OSS projects, spanning a range of applications. Due to their size, success and influence, the Linux operating system and the Apache Web Server are probably the most well-known, but hundreds of other projects are in widespread use.

Key to our interest is the fact that most OSS software is developed by distributed teams. Developers contribute from around the world, rarely or never meet face-to-face, and coordinate their activity almost exclusively by means of email and bulletin boards (Raymond, 1998; Wayner, 2000). What is perhaps most surprising about the OSS process is that it appears to eschew traditional coordination mechanisms (Mockus, Fielding, & Herbsleb, 2002). Participants describe the OSS development process as lacking many of the traditional mechanisms used to coordinate software development, such as plans, system-level design, schedules, and defined development processes (Herbsleb & Grinter, 1999b). Characterized by a globally distributed developer force and a rapid and reliable software development process, 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.

OSS development teams are a convenient research setting for studying distributed work, for several reasons. There are a lot of them, some have been outstandingly successful in meeting the challenges of developing large and complex software systems while others have not, and in many cases, records of their interactions and work products are publicly available. As well, these teams perform a common task, software development, in a somewhat similar fashion using similar tools, so by studying them we control somewhat for variations that would otherwise make distributed work practices difficult to compare. Finally, OSS development is an important phenomena deserving of study in its own right. Millions of users depend on systems such as Linux (and the Internet), but as Scacchi (2002) notes, "little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success".

In this paper, we report the early results of a project to study distributed work in Open Source Software development teams. We begin by answering perhaps the most basic question: what is the social structure of these teams? This question is important both in its own right and to assess how representative OSS is of distributed work in general. We also begin to explore (but by no means conclusively answer) the question: how is the structure of teams related to other variables of interest?

In the next section, we briefly review the literature on OSS team structure, before presenting our methods and data, analysis technique and results. We conclude with a discussion and suggestions for future research.

Literature review

We begin our study of OSS by investigating the basic structure of OSS teams. Several authors have described OSS development teams as having a hierarchical or onion-like structure (e.g., Cox, 1998; Gacek et al., n.d.; Moon & Sproull, 2000), as shown in Figure 1. At the centre are the core developers, who contribute most of the code and oversee the design and evolution of the project. For example, a study of the Apache Web server found that only about 15 developers contributed more than 80% of the code for new functionality (Mockus, Fielding, & Herbsleb, 2000). The core is described as small and exhibiting a high level of interaction, which would be difficult to maintain if the core group were large.

Surrounding the core are the co-developers. These individuals contribute sporadically by reviewing or modifying code or by contributing bug fixes. However, typically a core developer must review these contributions before they are accepted and further distributed. For example, during much of the development of Linux, Linus Torvalds personally reviewed and decided on all code submissions (Torvalds, 1999). The co-developers group can be much larger than the core, because the required level of interaction is much lower. Surrounding the developers are the users. A subset of users are active users, meaning that they use the latest releases and contribute bug reports or feature requests (but not code). Non-programmers may also contribute by writing documentation or translating the system. However, the majority of users (the outermost circle) are passive, meaning that they simply use the code, without making specific tangible contributions in return. Because most OSS projects are freely distributed, it is impossible to know precisely the size of the passive user population, so the border of the outer circle is indistinct.

It is important to note that the description of a project team illustrated in Figure 1 is based on a few case studies of large and prominent teams, such as the Apache Web server project. The model has good face validity and is consistent with research on other kinds of distributed groups (e.g., Wasko & Teigland, 2002). However, it has not been extensively tested and may not be representative of OSS projects in general. As well, understanding OSS team structure is important to assess if the lessons of OSS teams can be generalized to distributed work settings that are differently organized. In this paper, we assess how the proposed "onion" model holds up for a wider range of OSS projects. More specifically, we test the hypothesis that OSS projects exhibit a high degree of centralization.

Methods and data

To explore the structure of the teams, we employed social network analysis (SNA). SNA is a set of related techniques to analyze the structure of social groups based on characteristics of the network of interactions in the group. Because SNA may be unfamiliar to some readers, we will explain the data collection and analysis in some detail as we go along. The first step in the process is to collect interaction data. The analysis reported in this paper is of interactions related to the bug fixing process for OSS projects. Bug fixing was chosen because it provides "a microcosm of coordination problems" (Crowston, 1997). Furthermore, a quick response to bugs has been mentioned as a particular strength of the OSS process: as Raymond (1998) puts it, "given enough eyeballs, all bugs are shallow". As well, it is a process that involves the entire community, the core and co-developers as well as active users, and thus provides evidence to test the model presented above.

To create a sample of OSS projects, we selected from projects hosted by SourceForge, a free Web-based system that provides a range of tools to facilitate OSS development. At the time of our study, SourceForge supported more than 50,000 OSS projects on a wide diversity of topics. 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 study 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. Unfortunately, space does not permit a full listing of the projects, but Table 1 lists examples of the projects to give a sense of the sample. Those familiar with OSS may recognize some of these projects, which span a wide range of topics and programming languages.

To study the network of interactions around bug fixing, we collected data from the SourceForge bug tracking system, which enables 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 a trail of correspondence. To collect data for our analysis, we developed a spider program that downloaded all bug report pages for the selected projects. Data were collected 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. We obtained data on a total of 62,110 bug reports, an average of 509 per project. The median number of reports was 279, indicating a skewed distribution of bug report counts. The poster of bug reports and messages are identified by a SourceForge ID. We counted a total of 14,922 unique IDs, of whom 1,280 were involved in more than one project (one was involved in 8).

Downloaded bug report pages were parsed to extract interaction data, which were represented as a sociomatrix for each project. A sociomatrix is a standard data representation for a network analysis (Wasserman & Frost, 1994, p. 80). The matrix 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.

Two key issues in the application of SNA are the definition of an actor and of an interaction. In the SourceForge system, contributions to the system are identified by a unique SourceForge ID, which we used to identify actors. It is possible (and there is no way of knowing) that a single individual could utilize multiple SourceForge 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 want to maintain a single ID.

The definition of an interaction was more involved. For this analysis, we counted each message in the bug tracking system as one interaction from the sender of the message to the preceding identified sender (or to the original bug reporter). Bug reports that had no followup messages provided no interaction data. The arrows in Figure 2 show how two interactions were coded for this fragment of a bug report and messages. Note that messages are in reverse chronological order, so the response to the original report is actually the bottom message (not shown), the top message responds to the next, etc.

This definition of an interaction required several decisions about the source and destination of the interaction. Since messages are labelled with the sender's ID, identifying the source seemed straightforward. However, when a message is posted by a non-logged-in user, the sender is listed as "nobody". These messages, which constituted an average of 15% of the messages for a project (as low as 0% and as high as 50% for a few projects), are missing data for our analysis. 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 because the large number of such messages would have seriously biased our results. We experimented with two alternative approaches: 1) simply dropping the "nobody" interactions from the sociomatrix by deleting the "nobody" row and column; and 2) recoding 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). The second approach retains interactions between another individual and a "nobody" but at the cost of introducing a host of fictitious characters.

We also had to decide on the destination of each interaction. 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. Somewhat arbitrarily, we chose to code interactions as responses to the previous sender. Fortunately, the analysis approach chosen does not depend greatly on this choice.

Analysis

Once we had the sociomatrices, we performed several analyses to examine the projects' structures and to test our specific hypothesis. 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. Graph drawing turns out to be a surprisingly difficult problem involving a variety of tradeoffs; there is no single "correct" plot for a network. We experimented with several programs to draw plots, including NetMiner (Cyram, 2002) and Pajek (Batagelj & Mrvar, 1998). Figure 3 shows the interaction plot for a typical project, *openrpg*, created in 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.

To numerically test the hypothesis that projects have centralized structures, we calculated the network centralization score for each project. The calculations we report were computed using the SNA library in R (http://www.r-project.org/). 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). Because we were concerned that the dichotomization discarded data about the intensity of the interactions, we also calculated individual centralities using the raw interaction counts. In this case, actors' centralities are the total number of interactions.

Finally, 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. Netminer can create a plot grouping individuals in the network by their calculated centrality, with more central individuals at the centre and less central individuals on the periphery. Figure 4 shows such a plot for the project in Figure 3 (note that the number of bands in the plot was chosen by the researchers rather than as a result of the analysis). This plot confirms that the impression from Figure 3 that a small number of individuals in this project have high centralities, while the rest have low.

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. Wasserman and Faust (1994) do not give a formula for the theoretical maximum for raw interaction counts, so we calculated it for a star network assuming the total number of observed interactions had come from the central individual. This choice makes the centralization score the Gini coefficient for the inequality of the distribution of interaction counts. As a concrete example, the various centralization scores for the project in Figure 3 are listed in Table 2. Figure 5 is a histogram of the project centralization scores, calculated from dichotomized data, dropping "nobody" interactions.

Finally, to explore the potential of the centralization 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 centralization scores calculated in different ways, 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.

Discussion

Our first two findings are methodological. First, the choice of treatment for the missing-data "nobody" messages did not seem to make very much difference: the correlations between the centralization measures for the two approaches (highlighted in blue in Table 3) are both greater than 0.9. Second, while the centralization scores calculated from the raw interaction counts are lower (likely due to our conservative calculation of the theoretical maximum centralization), the correlations between these measures and the measures calculated from the dichotomized data (highlighted in purple in Table 3) were about 0.8 or higher. Because all the varied centralization measures seem to be measuring more-or-less the same thing, it seems simpler to use the conventional definition of centralization. The remainder of this section will therefore discuss only the centralization scores calculated from dichotomized data and without "nobody" interactions.

Our third finding is more substantive. To our surprise, our data indicate that OSS projects are not uniformly centralized, contrary to expectations. In fact, the calculated centralization measures display a considerable range, as shown in Figure 5. To check that this unexpected result was not an artifact of our calculations, we examined the interaction graphs for projects with high and low centralizations. An example of a highly centralized network, for the project *curl*, is shown in Figure 6 and of a large less centralized network, for *squirrelmail*, in Figure 7. These plots were drawn with the program Pajek. Figure 6 clearly shows an extremely centralized network, but Figure 7 presents a more complicated picture. There are still a large number of peripheral individuals, but the centre of the figure is made up of a dense network with no clear centre. (There are also nodes that appear to be isolates because Pajek did not plot self-interactions.) 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 0.56).

Fourth, the distribution of individual centrality scores, as indicated in Figure 4, is highly non-normal. A few individuals have a large number of interactions, while most have only a few. This property indicates that the pattern of interactions is not random, but may instead be "scale-free" (Albert & Barabási, 2002), similar to other networks such as the Internet, the Web and a wide variety of social networks. A scale-free network emerges as the result of a growth process in which new nodes preferentially attach to already-popular nodes, causing those nodes to become "hubs" of the network. However, it is not clear if this explanation works for our data, since our hubs are the ones deciding on the attachments (i.e., the developers choose which bug reports they will respond to rather than the bug reporters' choosing the developer to whom they will forward a report).

Our final finding is that the centralization scores are strongly negatively correlated with number of developers who contributed to the bug reports (highlighted in green in Table 3). Again, we were concerned that this correlation was a result of a bias in the computed centralization measure, but the plot of the relationship, shown in Figure 8, suggests otherwise: 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.

Conclusion

Clearly, much more remains to be done. Most immediately, there are several further analyses we plan to perform on the data we have already collected. First, it would be desirable to perform a more direct study of hypothesized project structure by examining the relative sizes of the network core and periphery. Formal models have been developed for core-periphery structures (Borgatti & Everett, 1999), but unfortunately they were not available in the analysis package we used for this paper.

A second extension is to study the evolution of project structures over time. Such an analysis would provide insight into how projects develop and change. As well, comparing the structure at different times would enable us to isolate the effects of changes in project leadership, which is a plausible alternative explanation for the observed decentralization of projects. Interactions might be initially centered on one set of leaders and later interaction on a different set, but plotting these interactions together would show a structure with multiple centres, even if the project was in fact always highly centralized.

Third, because some developers are involved in multiple projects, we would like to examine the pattern of interactions across multiple projects. We would expect the merged pattern of interactions to display clusters, but also some interconnections. However, because of the large number of individuals, this analysis will require considerable computation. We would also like to further explore the characterization of project structures, individually and collectively, as "scale-free", as doing so would allow us to connect our work to a growing body of research on such networks.

Further analysis will require additional data collection. A significant limitation of our findings is that we examined only a handful of projects in SourceForge that use the bug tracking system and that make bug reports public. We are comfortable that our selection criteria are appropriate for identifying truly active projects, but to go beyond these limits, we have started to examine other interaction data, such as patch reports and mailing lists, to see if they display similar patterns of interactions. Our research approach can also be extended to data from other project management systems, such as GNU's Savannah (http://savannah.gnu.org/ and http://savannah.nongnu.org/).

We are also beginning to exploring the relation between project structure and other variables of interest. In this paper, we reported the negative correlation between centralization and project size, as measured by the number of developers. We are in the process of collecting, from a variety of sources, additional data about projects and project performance that can be associated with the project structure. For example, identification of the relative size of the core and periphery would allow us to test the hypothesis that project success depends on having a strong active user community. Unfortunately, collecting data on performance (or even defining it) poses significant challenges. We are particularly interested in identifying measures of the bug fixing performance to go along with the bug-report centralization data presented here.

Finally, our initial interest in the Open Source projects was to better understand the nature of distributed work and how to successfully manage distributed teamwork. To really understand the nature of the work practices in Open Source teams will require us to examine not just the pattern of interactions but also their content, to observe elements such as coordination practices. Even so, these more detailed studies can be guided by the analysis we present here. For example, the scatter plot in Figure 8 suggests projects that will make particularly interesting comparisons for future work.

Bibliography

Ahuja, M. K., Carley, K., & Galletta, D. F. (1997). *Individual performance in distributed design groups: An empirical study*. Paper presented at the SIGCPR Conference, San Francisco.

Albert, R., & Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics, 74*, 47–97.

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 S. Kiesler (Ed.), *Distributed Work* (pp. 167–186). Cambridge, MA: MIT Press.

Batagelj, V., & Mrvar, A. (1998). Pajek: Program for large network analysis. *Connections, 21*(2), 47–57.

Bélanger, F. (1998). Telecommuters and Work Groups: A Communication Network Analysis. In *Proceedings of the International Conference on Information Systems (ICIS)* (pp. 365–369). Helsinki, Finland.

Borgatti, S., & Everett, M. (1999). Models of core/periphery structures. *Social Networks, 21*, 375–395.

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*, from http://sladhot.org/features/98/10/13/1423253.shtml

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

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.

Cyram. (2002). *NetMiner Webpage (http://www.netminer.com/)*. Retrieved 28 April, 2002, from http://www.netminer.com/

DeSanctis, G., & Jackson, B. M. (1994). Coordination of information technology management: Team-based structures and computer-based communication systems. *Journal of Management Information Systems, 10*(4), 85.

Drucker, P. (1988). The Coming of the New Organization. *Harvard Business Review*(3–15).

Finholt, T., Sproull, L., & Kiesler, S. (1990). Communication and Performance in Ad Hoc Task Groups. In J. Galegher, R. F. Kraut & C. Egido (Eds.), *Intellectual Teamwork*. Hillsdale, NJ: Lawrence Erlbaum and Associates.

Gacek, C., Lawrie, T., & Arief, B. (n.d.). *The many meanings of Open Source* (Unpublished manuscript). Newcastle upon Tyne, United Kingdom: Centre for Software Reliability, Department of Computing Science, University of Newcastle.

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. (1999a). Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*(September/October), 63–70.

Herbsleb, J. D., & Grinter, R. E. (1999b). Splitting the Organization and Integrating the Code: Conway's Law Revisited. In *Proceedings of the International Conference on Software Engineering (ICSE '99)* (pp. 85–95). Los Angeles, CA: ACM.

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

Krishnamurthy, S. (2002). *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*. Bothell, WA: University of Washington, Bothell.

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.

O’Leary, M., Orlikowski, W. J., & Yates, J. (2002). Distributed work over the centuries: Trust and control in the Hudson’s Bay Company, 1670–1826. In S. Kiesler (Ed.), *Distributed Work* (pp. 27–54). Cambridge, MA: MIT Press.

Orlikowski, W. J. (2002). Knowing in Practice: Enacting a Collective Capability in Distributed Organizing. *Organization Science, 13*(3), 249–273.

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

Robey, D., Khoo, H. M., & Powers, C. (2000). Situated-learning in cross-functional virtual teams. *IEEE Transactions on Professional Communication*(Feb/Mar), 51–66.

Scacchi, W. (1991). The Software Infrastructure for a Distributed Software Factory. *Software Engineering Journal, 6*(5), 355–369.

Scacchi, W. (2002). *Software Development Practices in Open Software Development Communities: A Comparative Case Study (Position Paper).*

Torvalds, L. (1999). The Linux edge. *Communications of the ACM, 42*(4), 38–39.

Wasko, M. M., & Teigland, R. (2002). The provision of online public goods: Examining social structure in a network of practice. In *Proceedings of the Twenty-Third International Conference on Information Systems* (pp. 163–172).

Wasserman, S., & Frost, K. (1994). *Social Network Analysis: Methods and Applications.* New York: Cambridge.

Wayner, P. (2000). *Free For All.* New York: HarperCollins.

Figures and Tables

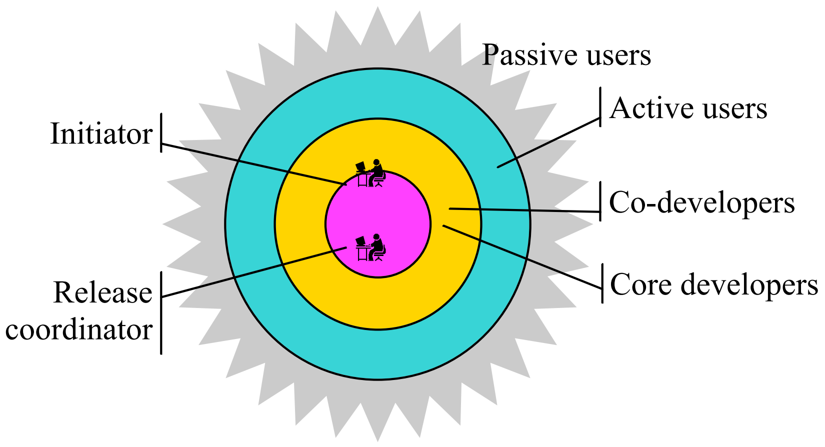
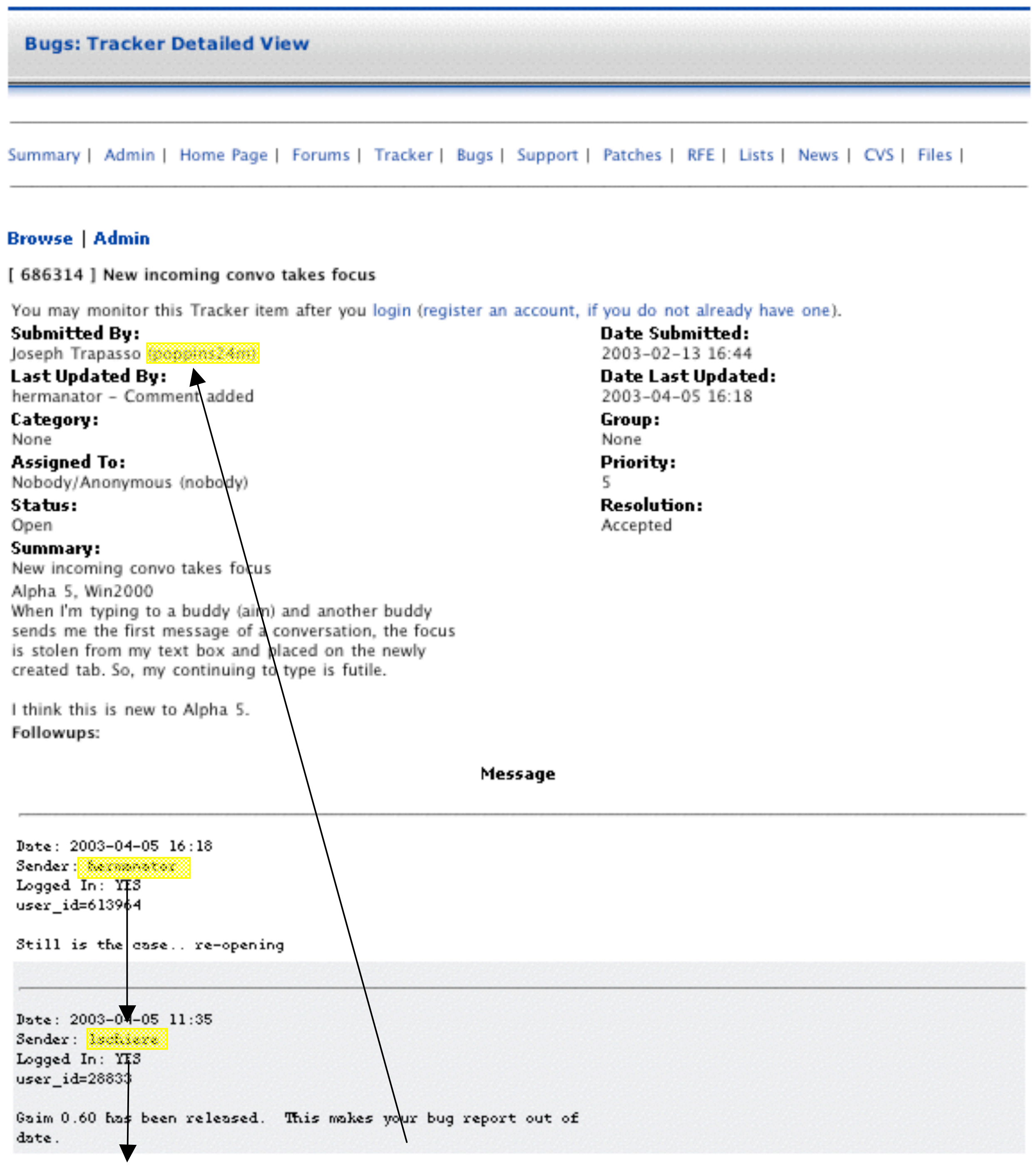


Figure 1. Hypothesized OSS development team structure.

Table 1. Examples of projects included in sample.

Project name	**Short description**
curl	Command line tool and library for client-side URL transfers.
fink	A source-retrieving package manager for Mac OS X.
gaim	A GTK2-based instant messaging client.
gimp-print	Top quality printer drivers for POSIX systems.
htdig	Complete world wide web indexing and searching system
jedit	A powerful text editor.
lesstif	LGPL'd re-implementation of Motif
netatalk	A kernel-level implementation of the AppleTalk Protocol Suite.
phpmyadmin	Handles the basic administration of MySQL over the WWW
openrpg	Play Role Playing Games in real-time over the Internet
squirrelmail	A PHP4 Web-based email reader.
tcl	Tool Command Language



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 (poppins24m)

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 followup showing coding of interactions (http://sourceforge.net/tracker/index.php?func=detail&aid=686314&group_id=235&atid=100235)

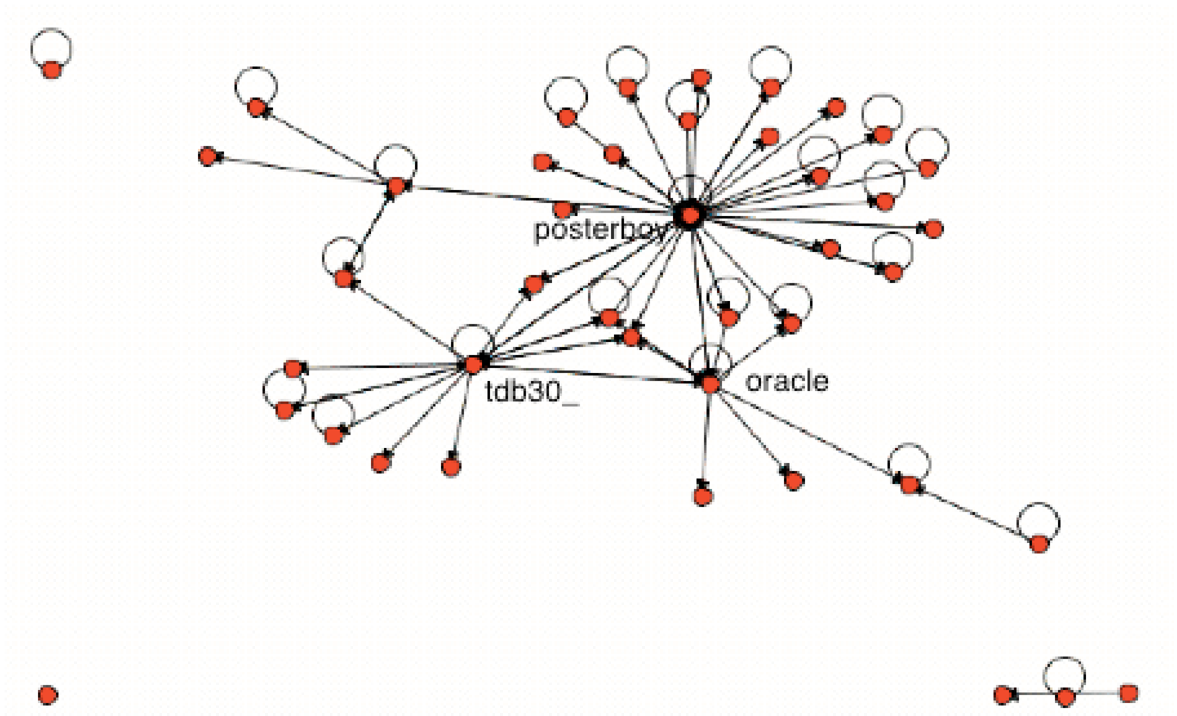


Figure 3. Plot of interactions for the openrpg project bug report data, created in Netminer.

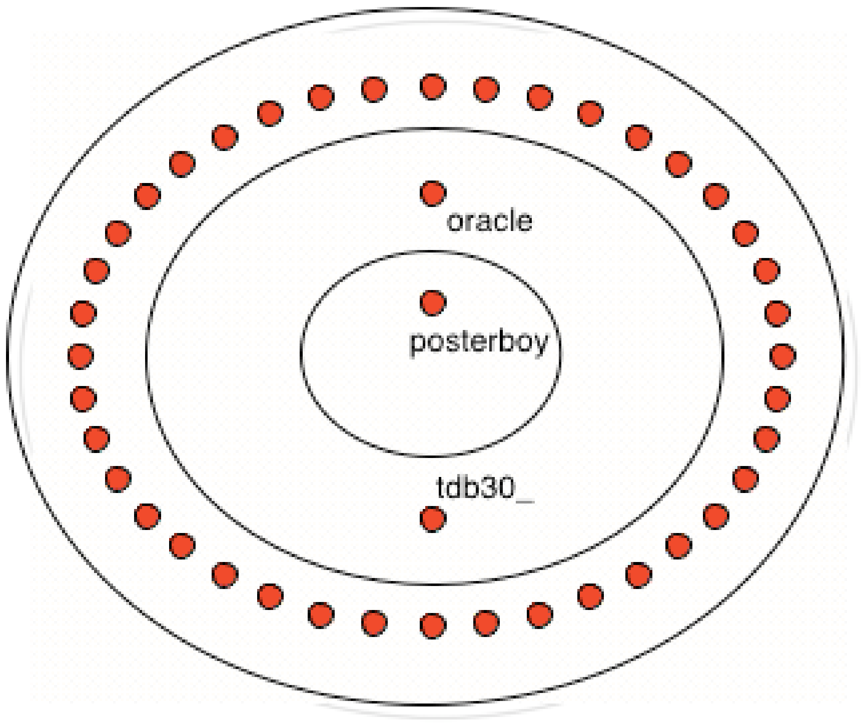


Figure 4. Centralization plot for the openrpg project bug report data, created in Netminer.

Table 2. Centralization scores for the openrpg project bug report data.

		Type of interaction data	
		Dichotomized	**Raw counts**
Treatment of missing data	**Nobody interactions dropped**	0.476	0.301
	Sender uniquely recoded	0.471	0.286

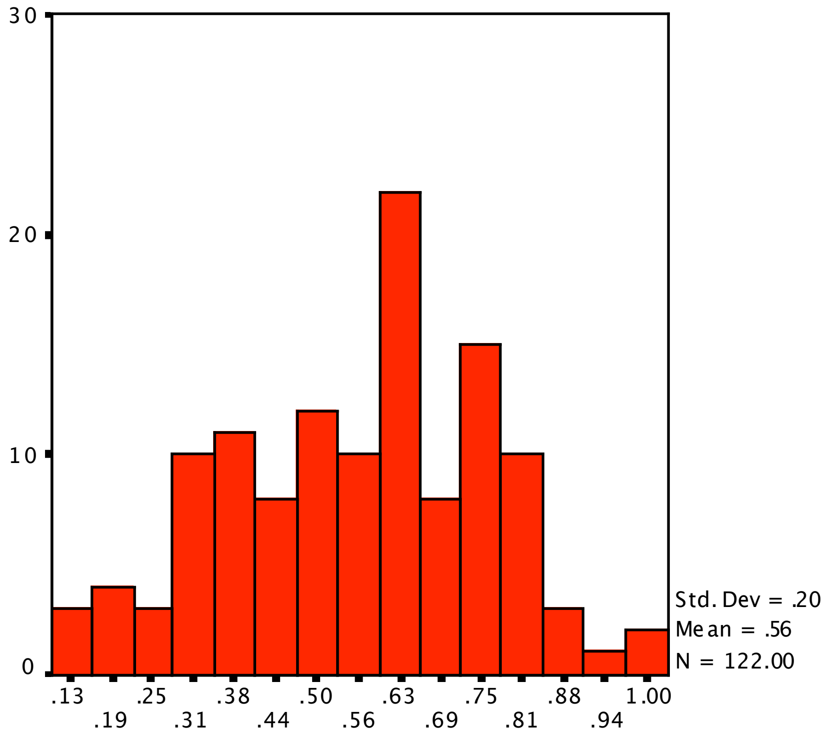


Figure 5. Histogram of centralization scores for projects. dichotomized data, dropping nobody interactions.

Table 3. Correlation among centralization measures and between centralization and project size

	B1O	**B1OE**	**B2O**	**B2OE**	**B1LN**	**B2LN**
Project centralizations						
B1O—outdegree, dichotomized, dropping nobody	1.000					
B1OE—outdegree, interaction counts, dropping nobody	0.791	1.000				
B2O—outdegree, dichotomized, unique nobodies	0.940	0.811	1.000			
B2OE—outdegree, interaction counts, unique nobodies	0.833	0.976	0.877	1.000		
Log number of individuals posting						
B1LN—dropping nobody	-0.371	-0.399	-0.423	-0.453	1.000	
B2LN—unique nobodies	-0.354	-0.330	-0.367	-0.375	0.918	1.000

N=122. All correlations are significant, p<0.01. Highlighting refers to discussion in text.

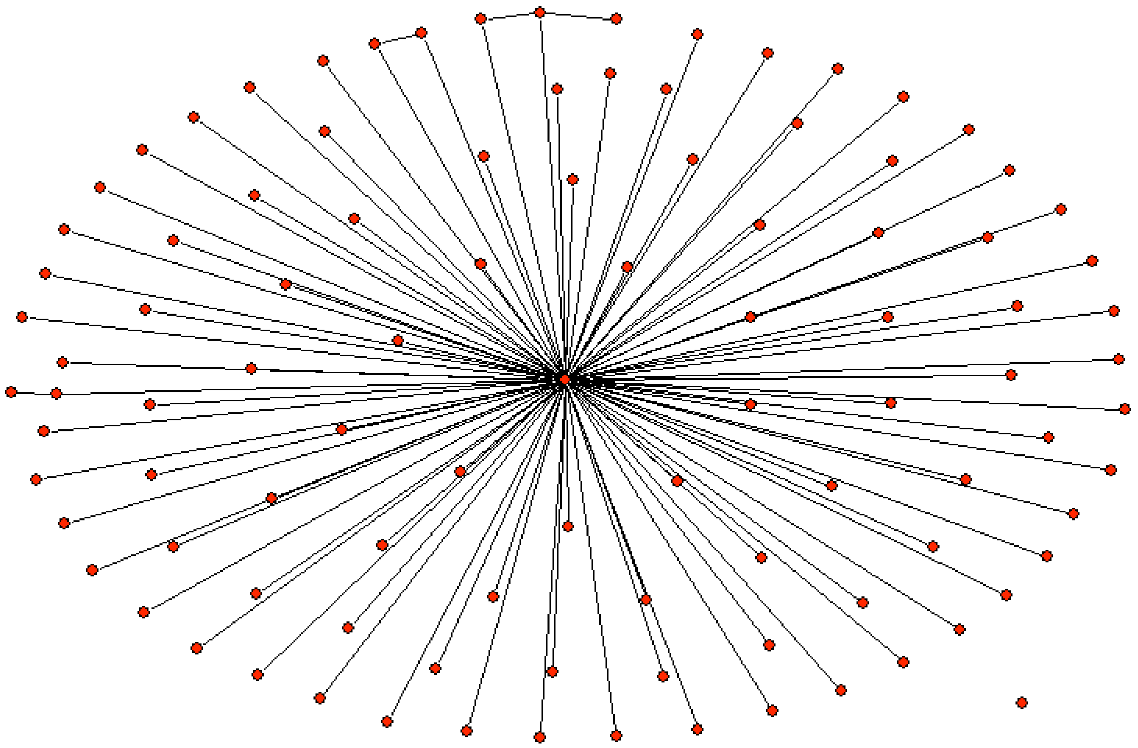


Figure 6. Plot of interactions for curl, a highly centralized project (centralization = 0.922).

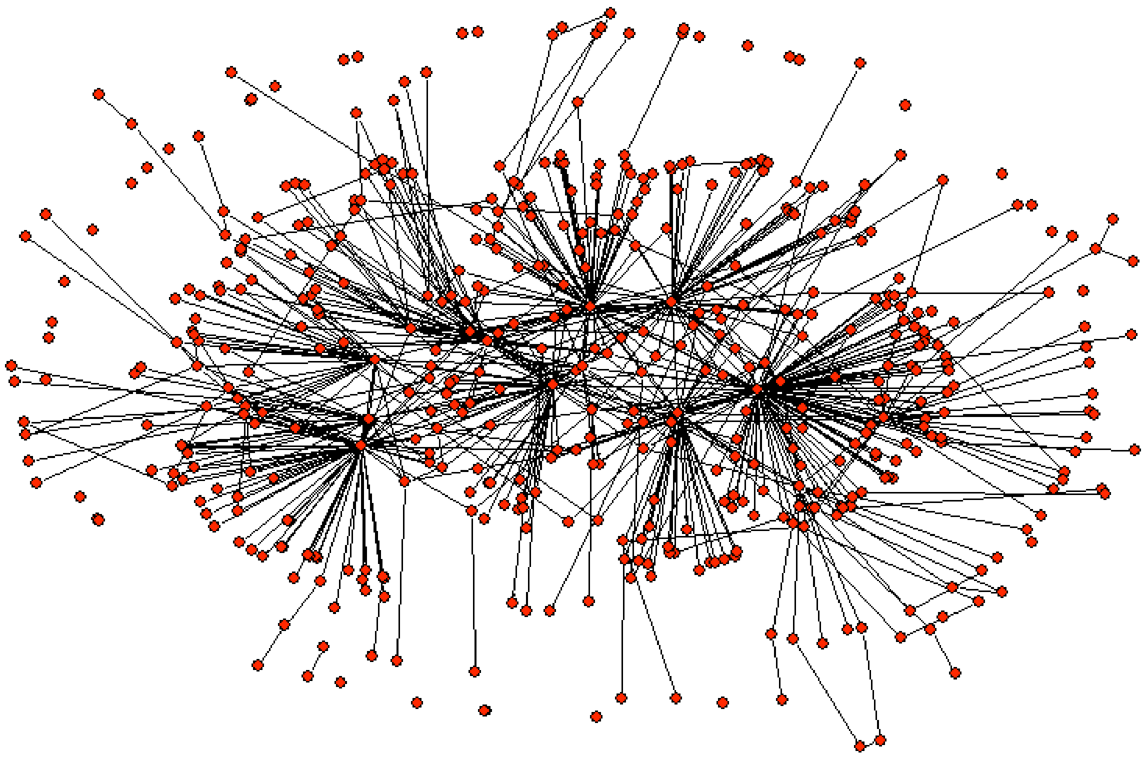


Figure 7. Plot of interactions for squirrelmail, a decentralized project (centralization = 0.377).

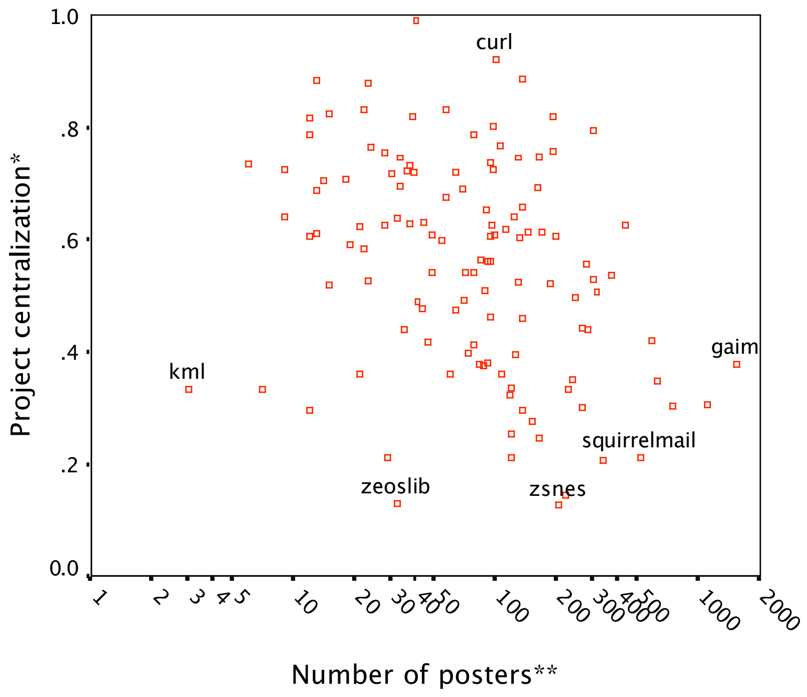


Figure 8. Plot of project centralization vs. project size as measured by number of posters, with extreme values labelled by project. r = –0.371, N = 122.
* Outdegree centralization, dichotomized data, dropping nobody interactions.
** Log scale, dropping nobody interactions.