The social structure of Free and Open Source software development*

Kevin Crowston and James Howison
School of Information Studies, Syracuse University
{crowston,jhowison}@syr.edu

Submitted for review in November 2004

Abstract

Metaphors, such as the Cathedral and Bazaar, used to describe the organization of FLOSS projects typically place them in sharp contrast to proprietary development by emphasizing FLOSS's distinctive social and communications structures. But what do we really know about the communication patterns of FLOSS projects? How generalizable are the projects that have been studied? Is there consistency across FLOSS projects? Questioning the assumption of distinctiveness is important because practitioner-advocates from within the FLOSS community rely on features of social structure to describe and account for some of the advantages of FLOSS production.

To address this question, we examined 120 project teams from SourceForge, representing a wide range of FLOSS project types, for their communications centralization as revealed in the interactions in the bug tracking system. We found that FLOSS development teams vary widely in their communications centralization, from projects completely centered on one developer to projects that are highly decentralized and exhibit a distributed pattern of conversation between developers and active users.

We suggest, therefore, that it is wrong to assume that FLOSS projects are distinguished by a particular social structure merely because they are FLOSS. Our findings suggest that FLOSS projects might have to work hard to achieve the expected development advantages which have been assumed to flow from "going open." In addition, the variation in communications structure across projects means that communications centralization is useful for comparisons between FLOSS teams. We found that larger FLOSS teams tend to have more decentralized communication patterns, a finding that suggests interesting avenues for further research examining, for example, the relationship between communications structure and code modularity.

*Currently under review (submitted November 2004). Until publication please cite as Crowston, Kevin and Howison, James (2004) "The social structure of Free and Open Source software" Syracuse FLOSS research working paper, available at http://floss.syr.edu

Introduction

There is a pervasive perception that Free/Libre and Open Source Software (FLOSS) development is different; different from proprietary, or traditional, or commercial or whatever other forms of software development it is that exist beside FLOSS. The ways in which FLOSS projects are perceived to differ are usually quite broad: they use different licenses, employ distinctive toolsets such as SourceForge and are largely composed of volunteers rather than paid employees, to name just a few. Dramatizing this perception, Eric Raymond coined his well-known comparison, "The Cathedral and the Bazaar" Raymond (1998a) which summarized these perceived differences neatly by drawing analogies to different types of organizations, each with distinctive patterns of decision making and communication.

Alan Cox, a senior Linux developer, introduces two more organizational metaphors, the "town council" and the "clique," in an essay published on Slashdot in 1998. Cox provides a "guide to how to completely screw-up a free software development project" by describing the early days of the Linux on 8086 project, "one of the world's most pointless exercises" and therefore one that has great "Hack Value." Cox writes that this obscurity meant that there were really only two or three people with both the skill and interest required, but also many "dangerously half-clued people with opinions—not code, opinions." These "half-clued" participants acted like a "town council" which created so much ineffectual noise that the core developers were lead to abandon the "bazaar model" and, using 'kill-files,' formed a "core team" which, Cox writes, "is a polite word for a clique." Cox argues that ignoring the "half-clued" was understandable but badly mistaken and ultimately caused the project to stall, not least because the real programmers were unable to help the wannabe programmers to learn to contribute usefully and thus change the unproductive "town council" into a productive project.

Effective FLOSS projects, then, are organized like "bazaars" but ideally not like "town councils" or "cliques" and certainly not like teams building a "cathedral." Each of these organizational metaphors place communication at the center of their understanding of FLOSS development and its distinctiveness: The energetic hubbub of the bazaar contrasts with the solemn controlled ceremonies of the cathedral, the ineffectual carping of a town council and the insular exclusion of the clique. Not only, then, are FLOSS projects expected to differ from proprietary development in licenses, tools and motivations for their work but also in their communication style.

Even those that are not entirely comfortable with these characterizations of the organization of FLOSS development (eg Krishnamurthy (2002)), rarely go so far as to suggest that there are not clear differences between FLOSS and proprietary development. The emphasis on comparison with proprietary development, combined with the fact that most research on FLOSS has been case studies of particular projects, has so far allowed the perception that there is a distinctive FLOSS organizational pattern, and set of practices, to go largely unquestioned.

This is particularly problematic because, as we show below, many of supposed strengths of FLOSS development are closely linked to the unique communications or social structure of the projects. The practicioner-advocate literature appears to be based on its authors' experiences with projects that are important, yet quite limited in number. It is assumed, rather than argued, that these particular desirable social structures, and the strengths they impart, will naturally be available to and be found in FLOSS projects in general. If that were not to be the case. and it is far from

obvious that it is, much of the argument for "going open" would need to be reframed. Certainly a greater community effort to intentionally create these desirable communication structures in FLOSS teams would be called for.

The question of consistency in social structure is also important to researchers examining FLOSS because it speaks directly to the boundaries of the phenomenon to be studied. If consistency in structure is found then credence is lent to the idea that it is sensible to study FLOSS as a phenomenon apart, but if consistency is not found credence is lent to the idea that FLOSS practices should be studied as part of a broader phenomenon of change in software engineering practices, such as the move towards distributed and agile development.

This paper, therefore, questions the assumption of consistency in social structure through an empirical examination of the communications structure of FLOSS projects, one of the central themes raised in the metaphors above. Specifically, for reasons discussed below, we assessed the consistency of communications centrality in bug-fixing processes to see if there was consistency across FLOSS projects.

We first briefly review the concept of social structure and introduce communications centralization as a relevant measure. We then examine both the practicioner-advocate literature and the academic literature that has begun to address the social structure of FLOSS projects before outlining our method, presenting our results and drawing conclusions.

Understanding Social Structure

Investigating social structure is a useful way to understand team practices. It allows research to begin to investigate questions of coordination, control, socialization, continuity and learning—all topics of great interest in studies of collaborative groups. Social structure has also been gaining interest in software engineering. As Scacchi writes, "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." Scacchi (2002) Software engineering has increasingly come to understand that there are predictable relationships between the structure of code and the social structure of the development team; better understanding of social structure can improve development planning.

Social structure is also useful in risk management because it allows an assessment of questions like, "who is so crucial to the project that their withdrawal could threaten its existence?" That this need for risk management is not limited to the proprietary community was confirmed in an interview we conducted with a member of the Apache Foundation's incubator team at ApacheCon 2003. The Apache foundation is a prestigious umbrella organization for teams developing FLOSS and has created an incubator to ensure that the projects seeking to join the Foundation are of sufficient quality and longevity[1]. The incubator team indicated that they were concerned that overly heavy reliance on a small number of (possibly corporate funded) developers was a major threat to the sustainability of the project and thus to the suitability of the project for Apache incubation. Grasping the social structure of the project would assist in such risk assessments.

Inquiry into social structure is by itself, of course, not sufficient for a full understanding of these

questions. For that it must be combined with qualitative and interpretivist analysis of the substance of code and communication. Inquiry into social structure is simply a beginning towards understanding. However, as in this paper, this journey can provide useful insights through comparisons and the identification of common or odd patterns, or changes in patterns, that are worth investigating in more detail.

It is helpful to consider three aspects of social structure: individuals, their actions and their interactions. By considering individuals we can obtain a picture of the group: is it small, large, constant, growing or shrinking? Further insights can be gained by understanding the patterns in the actions of those individuals: who is doing the work? Is it evenly spread between the project members, or concentrated in sub-groups? Is there specialization, do specific roles exist or do project members all contribute in the same way?

Inquiry need not, however, stop with these questions because when humans collaborate they interact with each other. Research can therefore focus on patterns of interactions between individuals engaged in action: who talks to whom and how often? Where and how do they talk? Are there individuals or sub-groups who are central to the project because communications flows through them, or do the project's members most often talk directly to one another? It seems likely that interaction is where we must look for questions of coordination, control and social learning. Inquiries into patterns of interaction can be undertaken through social network analysis (SNA) and this is the approach adopted in this paper.

One measurement of social structure is centrality. In general, individuals who have more interactions are more central than those who have less. Central individuals, therefore, are highly active and provide a crucial linking point for a social network. We provide more detail on the analytical definition of centrality in our Analysis and Findings section below.

In the context of FLOSS development, it is useful to distinguish between two forms of centralization which are implicit in, and will structure our review of the literature below: development centralization and communications centralization. In terms of our classification above, development centralization is an action measure while communications centralization is an interaction measure. Development centralization refers mainly to the writing of code. Projects highly centralized on this measure would have a small core of regular code committers, whereas more decentralized projects would have code written by a greater proportion of the people involved in the project. There is no doubt that development centralization is an important aspect of FLOSS structure. Indeed, without the production of code there would be nothing to study. However there are numerous practices which lead up to the code submission process that we identify as important in understanding the social structure of FLOSS teams and especially important in understanding them as virtual teams. These interactive practices include coordination, socialization and group learning. The structures implicated in these practices can best be described through a study of communications centralization.

Communication centralization refers to centralization in the communications between project members that are found in, for example, email, bug-reporting systems and instant messaging. Projects with high communications centralization would have a small number of people who speak to a large number of people who don't speak amongst themselves but only back to the small central group. By contrast, a decentralized communications network would exist in a project when most projects

members speak to a large number of other project members, not just a limited group. Communications centralization is one concept that can assist in measuring the communication structures suggested by the metaphors of cathedrals, bazaars, town councils and cliques.

We now review the way the concept of social structure and centralization has been discussed in the literature researching FLOSS development, both practitioner-advocate and academic. As we shall see the majority of studies of social structure in FLOSS have focused on development centralization and have not yet adequately addressed communication centralization.

Social structure in practitioner-advocate literature

The practitioner-advocate literature claims that particular aspects of social structure reflect and facilitate the advantages of FLOSS production. These strengths are often cited in arguments for "going open" or for the superiority of FLOSS development. It is generally assumed that they are available to or should be found within any FLOSS project.

Raymond (1998a) introduces what he dubs 'Linus' Law': "Given enough eyeballs, all bugs are shallow." He goes on to report that Linus Torvalds, the founder of Linux, believes that "the person who understands and fixes the problem is not necessarily or even usually the person who first characterizes it." He writes that Linus says, "Somebody finds the problem ... and somebody else understands it." Though the focus of Linus' law is on absolute numbers of participants, developers, co-developers and active users, 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. This 'law' is often relied upon when arguing that FLOSS development leads to more secure and less buggy products.

It is also claimed that FLOSS projects have an open development process in which planning is decentralized. 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." The image is one that suggests that FLOSS projects would would tend towards decentralized communications structures. This structure is placed in marked contrast to the top-down 'cathedral' of proprietary software engineering centered on the "architect." The requirement for such decentralized communications structures, if the notion of an open development process is to be borne out, is increased by its interaction with the norm against project forking (Raymond, 1998b). If projects are not to be forked, yet the project is to be open to user influence, then decentralized structures, which decrease opportunities for control and increase opportunities for input, must be expected.

Alan Cox addresses some of these issues in his analysis of the Linux on 8086 project (Cox, 1998) discussed in the introduction. Recall that he presents, "a guide to how to completely screw-up a free software development project" and argues that the developers in this project (including himself) "walled themselves off" from the participants, forming a "clique" not a bazaar. They did this by using a 'kill-file,' which meant that the clique did not read all the messages on the mailing list—those

from persons in the 'killfile' go straight to the trash, unread. These actions would have resulted in a centralized network because members of the clique would have had intensive interactions only amongst themselves: other individuals would send messages to those in the clique, but not vice versa. The flow would have been one-way towards the centre. This 'kill-file' attitude was, Cox argues, an understandable but ultimately wrong response. Cox argues that no matter how important it is to reduce distractions from 'town councillors,' 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 (but keep discussions focused on code that exists rather than merely ideas).

Simultaneously, however, the practitioner-advocate literature also praises social-structures which appear to be centralized. 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. The importance of ownership suggests that a certain centralization is crucial to the reputation or gift economy that has been a successful driver of FLOSS production. Yet to complicate this picture further, Raymond (1998b) suggests that 'ownership' is coupled with a norm amongst central figures to "speak softly." Using the distinction between development and communications centralization introduced above this suggests that high development centralization might not be reflected in high communications centralization.

The picture emerging from this literature clearly highlights social structure as an important aspect of FLOSS development practices. The practitioner-advocates appear to be in agreement that FLOSS projects are effective in part because of the way they are structured and that this structure is, in some way, new. That their discussion appears to describe possibly conflicting structures may indicate the 'bluntness' of measures like network centralization or reflect, as we feel it might, the tendency to discuss certain personally known anecdotes for advocacy rather than an attempt to characterize all FLOSS projects. Certainly it is worth recognizing that the practitioner-advocates suggestions are based on a necessarily limited, even if deep, experience of FLOSS projects and might not be true across all projects. That the practicioner-advocates choose to continually refer to FLOSS social structure shows that they intuitively consider it important and clearly invites confirmatory research which attempts to assess whether there is, in fact, consistency in FLOSS social structure.

Social structure in academic studies of FLOSS

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 *et al.*, 2001; Moon and Sproull, 2000; Mockus *et al.*, 2002). 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

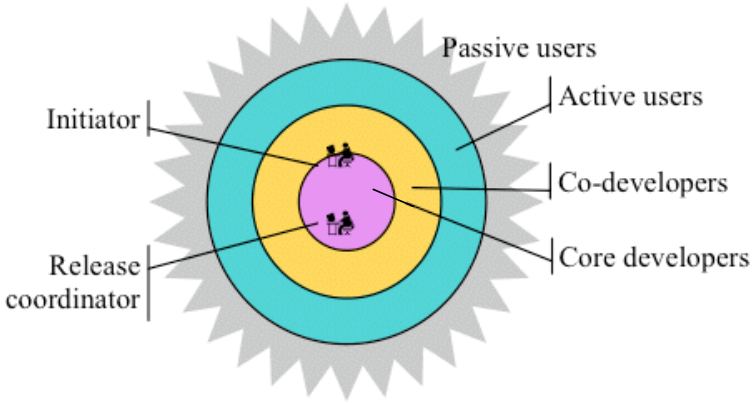

Figure 1: A synthesised FLOSS development team structure

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. It is an open question (which our study doesn't answer) as to whether these projects, which are clearly centralized in development, are also centralized in their communications structure. It is also an open question as to whether scholarship on FLOSS practices should take a great interest in these projects or read a great deal about the effectiveness of FLOSS practices into their apparent failure.

Together the case studies of FLOSS projects suggest a hypothesized model of FLOSS development as having a hierarchical or onion-like structure, which we have attempted to characterize in Figure 1 (Cox, 1998; Gacek *et al.*, 2001; Moon and Sproull, 2000; Mockus *et al.*, 2002). The focus of these studies has largely been on the contribution of code and they therefore have largely discussed development centralization. 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 speak on the project's lists or forums.

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." (Mockus *et al.*, 2002, p. 329). We have tried to illustrate this summary in Figure 1.

The case studies cited above only briefly touch on the topic of communications centralization.

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, suggesting centralization in communication patterns, 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 don't examine patterns of communication which follow up these reports and which show the FLOSS development process in action. It is these interactions which are the focus of our study below.

Our study

To address the question of whether FLOSS projects exhibit consistency in their social structure we chose to examine the network centrality of communication during the bug-fixing process. Our findings are therefore limited to an understanding of communications structure in the bug-fixing process. 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. The collaboration of bug-fixing potentially produces rich interactive collaborations. Finally, as described above, bug-fixing is the site of claims of effectiveness made for FLOSS projects.

To explore the structure of the teams we selected projects from SourceForge and downloaded project and bug-database data using web spiders. We then extracted interaction data from each bug-report to create interaction matrices. These were analyzed using social network analysis (SNA).

As befitting a study of FLOSS, the majority of our analysis was performed using free software, the spidering and parsing of SourceForge data was done with `perl` scripts and the social network analysis largely using the `sna` package Butts from the `R-project`(R Development Core Team, 2004). The only non-free software we used was network graphing software to produce some illustrations. As befits an investigation of FLOSS, the scripts and data (both raw and parsed) used in this analysis are available from our project website [2] and we have reported our experience in this large scale FLOSS data collection elsewhere (Howison and Crowston, 2004) [3].

Data

To create a sample of FLOSS projects, we selected from projects hosted by SourceForge, a free Web-based system that provides a range of tools to facilitate OSS development[4]. At the time of our study, SourceForge supported more than 50,000 OSS projects on a wide diversity of topics (the current figure is around 88,000). 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 or do not use the SourceForge Tracker system, which was the source of our data.

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 chose 7 developers as a criteria because we are interested in team interactions, not individual projects, and communicative interactions. We chose 100 bugs as a criteria to provide sufficient interactions to give a robust picture of the social network. Together these two criteria also meant that the projects were relatively active.

Project name	Short description
curl	Command line tool and library for client-side URL transfers.
python	Scripting language similar to perl and ruby.
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.

Table 1: Examples of projects included in sample

Somewhat surprisingly 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. Because we were concerned that some projects do not host all their bugs on SourceForge we examined a sample of the chosen project's instructions on how to submit bugs and satisfied ourselves that the SourceForge was a main repository. For example, python's homepage has a link "Report bugs" that links directly to the SourceForge tracker[5].

We based our analysis on the SourceForge bug tracking system. This system enables users and developers to report and discuss bugs. As shown in Figure 4, a bug report includes basic information about the bug that can be followed up with a trail of correspondence. Our spider program 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 124 projects. Four projects were found to have fewer than 7 participants in the bug forums (even though their homepage listed more than 7 developers) and were therefore excluded from the analysis.

From the 120 projects that we did analyze we obtained data on a total of 61,068 bug reports, an average of 509 per project. The median number of bug-threads per project was 286, and the standard deviation was 599 indicating a skewed distribution of bug report counts, that is confirmed by the illustrated distribution in Figure 2. Note that these numbers differ from the number of bug-reports reported on the website because we only counted bugs in which at least one reply had been posted.

The posters of bug reports and messages are identified by a SourceForge ID. We counted a total of 14,922 unique non-anonymous IDs, of whom 1,280 were involved in more than one project (one was involved in 8). The average project size, as measured by a count of unique SourceForge IDs posting to the tracker, was 140 posters, but again the distribution was highly skewed with the smallest having only 9 posters, and the largest having 1521 posters and the median project having 86. The distribution of project sizes is shown in Figure 3 where the largest projects in the tail were

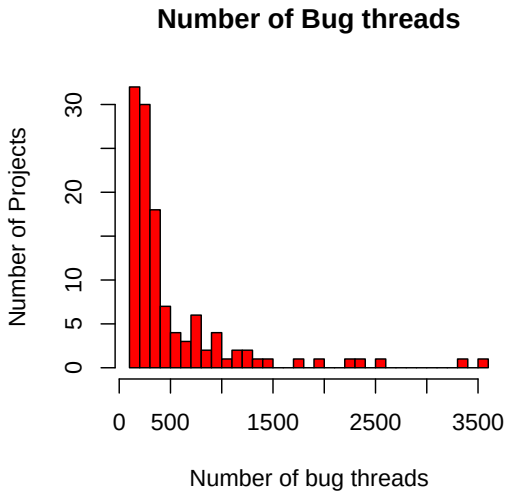

Figure 2: The distribution of the number of bug-threads for each project.

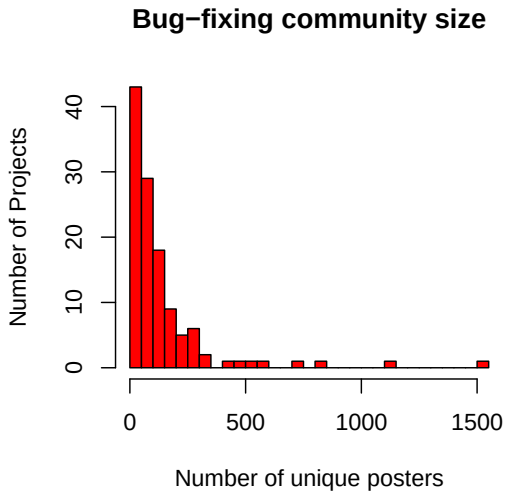

Figure 3: The distribution of project size measured number of posters to the bug-tracker.

`gaim`, `python` and `dcplusplus`. As should be expected there is a high correlation between project size and the number of bug threads in the SourceForge tracker ($r = 0.84$). This accords with the intuitive understanding that larger communities find and communicate about more bugs.

Analysis and Findings

Since SNA analyzes interactions between actors, 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 that a single individual could utilize multiple SourceForge IDs or that multiple individuals could share one. However, we believe it is unlikely that many individuals do maintain multiple accounts both because it would be cumbersome and there would no incentive to do so. Indeed, because reputation accrues to an ID, we believe that most individuals will want to maintain a single ID. Therefore we used the SourceForge ID as our definition of actors in the system.

The definition of an interaction was more involved. For this analysis, we counted each message associated with a bug in the bug tracking system as one interaction from the sender of the message to the preceding identified sender (starting with the original bug reporter). Bug reports that had no followup messages provided no interaction data while most bug reports provided multiple interactions. The arrows in Figure 4 show how two interactions were coded for this fragment of a bug report and messages. Note that messages appear on the page 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

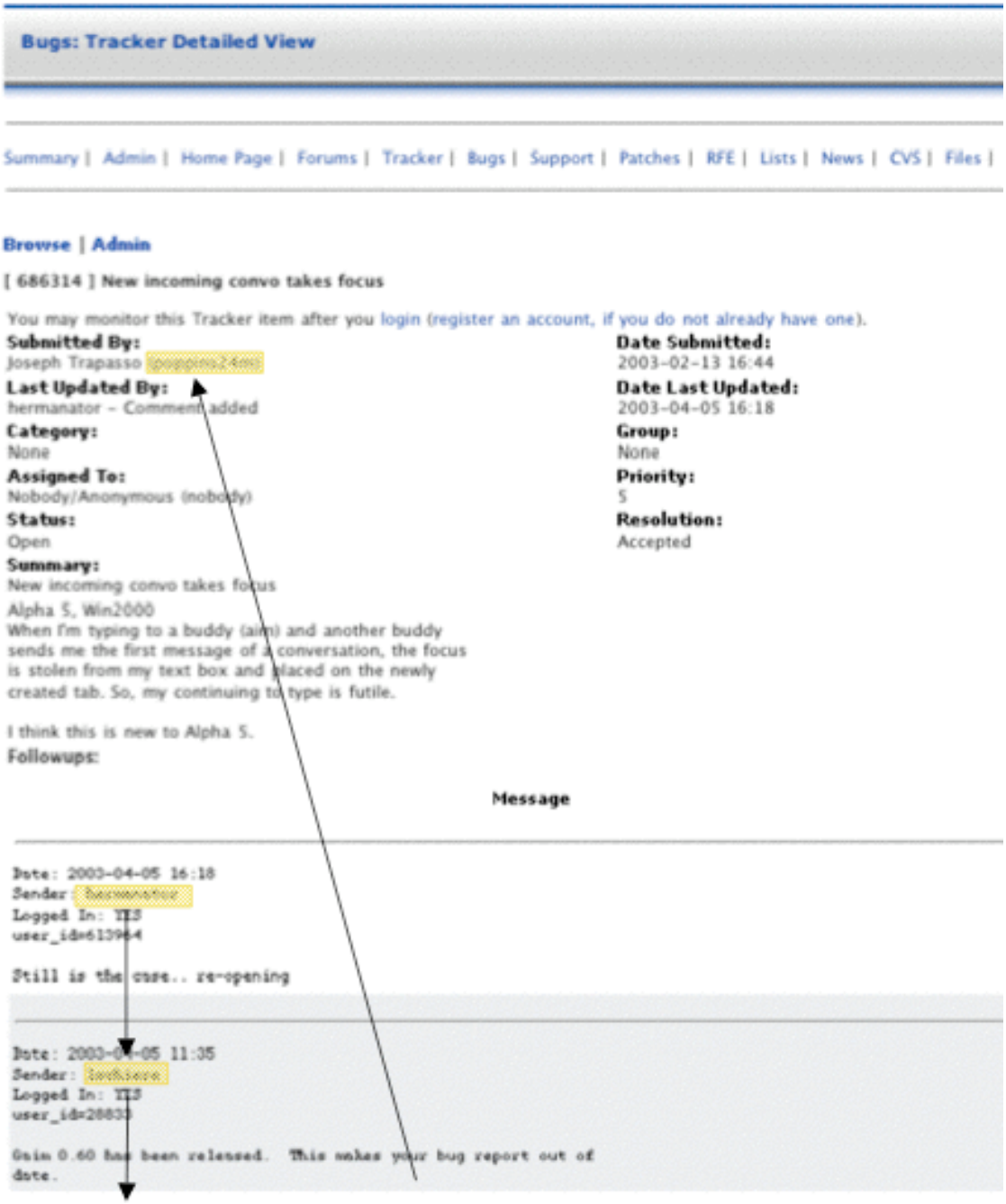

Figure 4: Example SourceForge bug report and followup showing coding of interactions

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 23 percent of the messages for a project (as low as 0 percent and as high as 60 percent for one project) are missing data for our analysis. We considered several alternate 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. In projects with high missing data, "nobody" would have been the most central actor. We experimented with two alternative approaches: 1) simply dropping the 'nobody' interactions (taking care not to create fictitious interactions in the process); 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 possibly introducing fictitious characters. We found that the different treatments did not materially alter our results, so to simplify our presentation we present analyses based on data from which the missing data has been dropped.

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 the previous messages and sometimes to the original poster. Unfortunately, the true destination is difficult to determine mechanically. Based on a sample reading of bug-messages we chose to code interactions as responses to the sender of the previous message.

Once we had extracted the interaction data we first plotted the interaction graphs for a selection of projects to visualize the interactions and get a sense of the data. We experimented with several programs to draw plots, including `NetMiner` (NetMiner Webpage) and `Pajek` (Batagelj and Mrvar, 1998). Figure 5 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). However, in these plots the distance between nodes is an approximation of the strength of the ties, using the spring embedding algorithm of Kamada and Kawai (1989). The plot in Figure 5 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).

The `Netminer` program 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 6 shows such a plot for the project in Figure 5 (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 5 that a small number of individuals in this project have high centralities, while the rest have low.

To numerically assess whether the projects had consistent social structures, we calculated the network centralization score for each project. The calculations we report were computed using the `sna` library in the `R` statistical package.

There are many different definitions of centrality in the literature (Wasserman and Frost, 1994, Chp. 5) and the choice between measures is based on the substantive nature of the interactions.

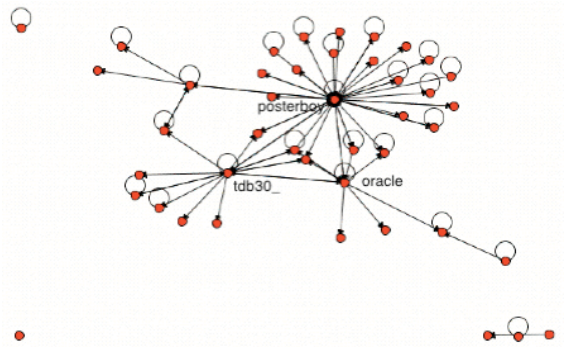

Figure 5: Plot of interactions for the openrpg project bug report data, created in Netminer

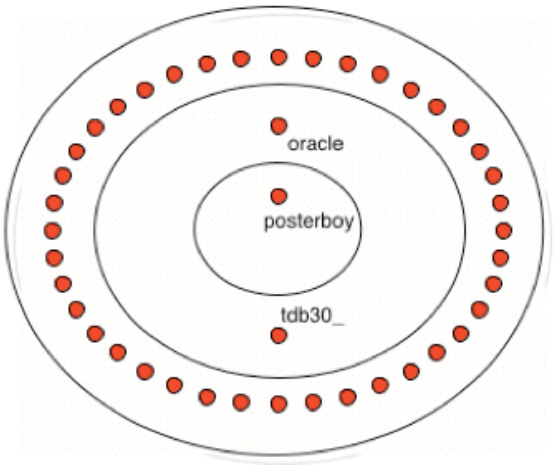

Figure 6: Centralization plot for the openrpg project bug report data, created in Netminer

We decided to use the most basic definition, which is based on degree: individuals who receive or send more messages 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 bugfixing process than one who posts fewer. 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 use out-degree centrality because of our interest in identifying individuals who contribute to a broad range of bug reports.

These individual centrality scores can be used to characterize group centralization, a property of the whole network which is defined in terms of inequality in centrality scores: 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 and Frost, 1994, p. 176). An illustrative example of a fully centralized network is a star-shape where there is one individual who interacts with everyone and in which everybody else only interacts with that central individual (giving 'perfect' inequality in centrality scores). At the other end of the spectrum a fully decentralized network would exist when each network node interacts with other nodes to the same degree[6]. Rather than a star-shape this network is a 'thicket' of interactions in which each person's centrality score is the same. Figure 7 depicts these two ideal-type networks.

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. The usual calculation of degree centrality is based on dichotomous data (i.e., communication vs. no communication). We dichotomized the interactions 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 full calculations are available in the scripts on our

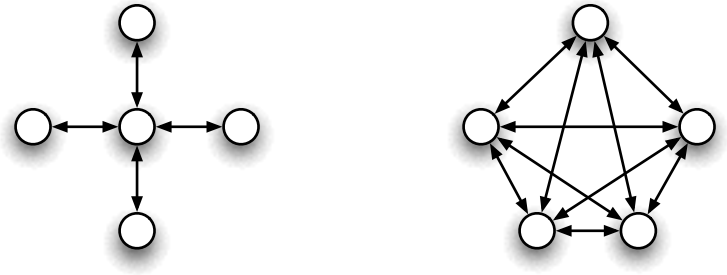

Figure 7: The 'star' on the left shows full centralization (cent = 1.0) while the 'thicket' on the right shows full decentralization (cent = 0.0).

website.

The distribution of project centralization measures

Figure 8 is a histogram of the project bug interaction centralization scores, calculated from dichotomized data, dropping 'nobody' interactions. Our data indicate that, for OSS projects engaged in the bug-fixing process, communication structures are not uniformly centralized nor are they uniformly decentralized. Rather, the calculated centralization measures display a considerable range. The most centralized project had a centralization of 0.99, the least had a centralization of 0.13, the mean centralization was 0.56, and the standard deviation was 0.20. The median value (0.58) was close to the mean showing an un-skewed distribution. Indeed the Shapiro-Wilk test (Royston, 1982) for normality showed that the distribution cannot be distinguished from a normal distribution (W= 0.9847, p-value = 0.1934 where low values of W would indicate deviation from normality and the high p-value means that the hypothesis of normalcy cannot be rejected).

To check that this 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 9 and of a large less centralized network, for `squirrelmail`, in Figure 10. These plots were drawn with the program `Pajek` (Batagelj and Mrvar, 1998). Figure 9 clearly shows an extremely centralized network, but Figure 10 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.

Although not reported here, we examined communications centralization on developer mailing lists for the 52 projects where they were available. The results showed similar results with a wide distribution of centralization scores.

Our first finding, then, is that our data demonstrate that the centralization of OSS projects engaged in bug-fixing is in fact widely distributed, with a few highly centralized projects, a few decentralized and most somewhere in the middle (the mean centralization is 0.56 and the distribution cannot be distinguished from a normal distribution). This finding is strengthened, not weakened, by its reliance on data only from SourceForge: if there was a systemic bias introduced by using data from teams that all use the SourceForge tools, it should be to promote similarity, making our finding

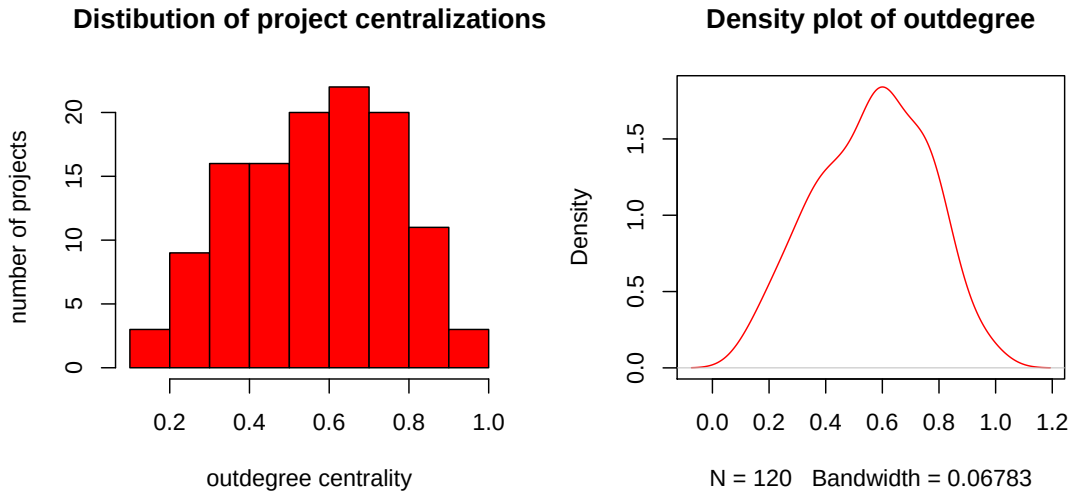

Figure 8: Histogram of communication centralization scores for projects engaged in the bug-fixing process. Dichotomized data, dropping nobody interactions. Mean = 0.56, MD = 0.58, SD = 0.20

of wide variance more unlikely and thus stronger. It does not seem true to say that there is a consistent structure that distinguishes communication in the bug-fixing process of FLOSS projects.

We must, however, acknowledge the possibility that our analysis has disguised complex patterns in the data. An acknowledged limitation of our analysis is that it groups all interactions over time into one network. This raises the possibility that our statistics are artificially decentralized for projects that are, in fact, highly centralized but have changed leaders. Such a project would show up as much more decentralized than they are in day-to-day practice. We visually examined plots to assess whether this has happened and did not find evidence of such a pattern. We are preparing a dynamic analysis to fully assess this interesting possibility which would indicate a qualitatively interesting episode to study.

Relationship between project centralization and project size

The very variance of the project centralization scores, however, make our centralization measure of use in distinguishing between FLOSS projects. We were therefore interested in the relationship between project centralization and project size because the group collaboration features which we are interested in, such as coordination and learning, often face significant problems in scaling from small to large groups. Given these interests the number of individuals involved in the bug discussions was the appropriate measure of project size. We calculated Pearson correlations coefficients between the centralization scores and the number of developers and active users who contributed messages to the bug report tracking system. 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. We found that the centralization scores are negatively correlated with number of developers and active users who contributed to the bug reports. The

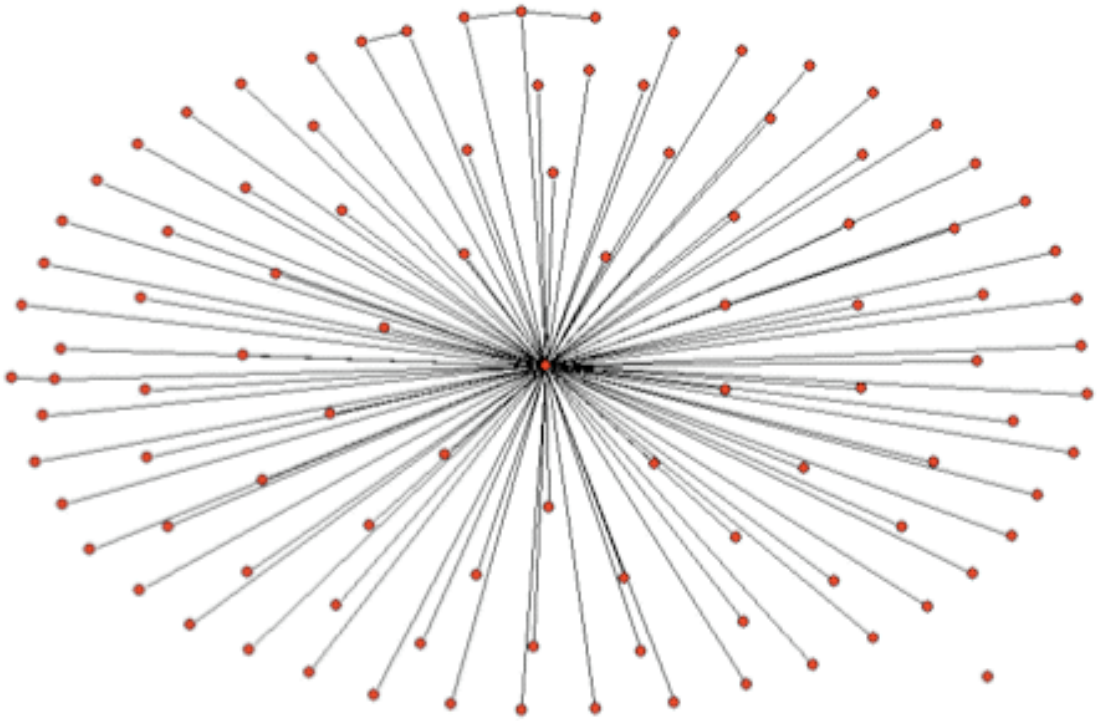

Figure 9: Plot of interactions for curl, a highly centralized project (centralization = 0.922)

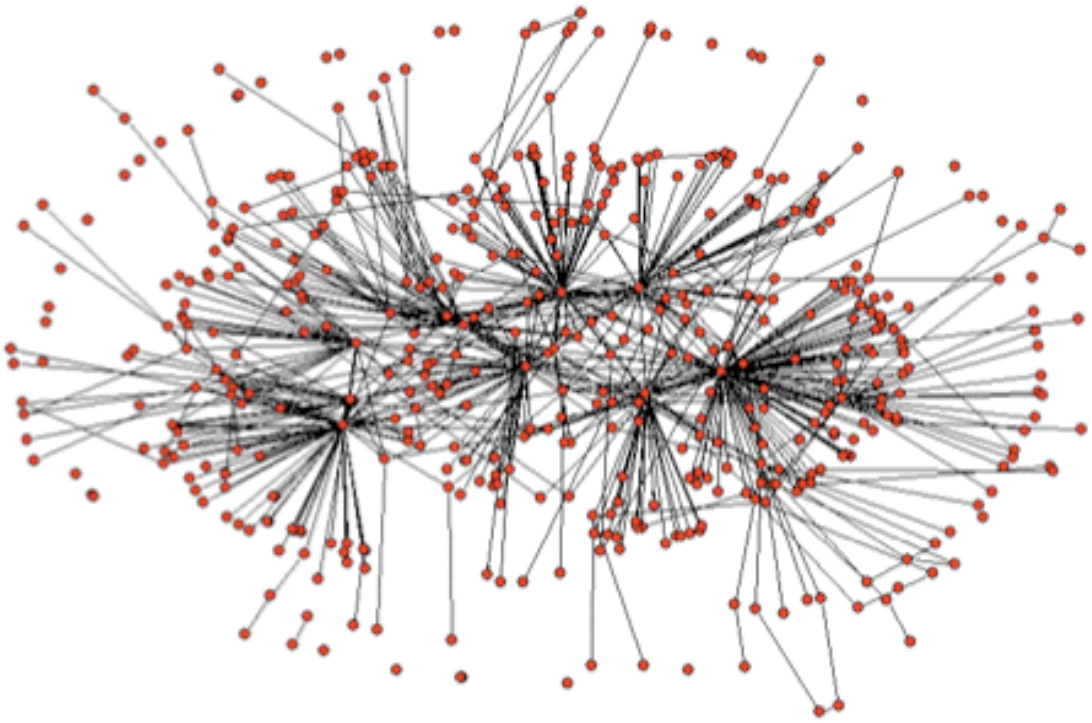

Figure 10: Plot of interactions for squirrelmail, a decentralized project (centralization = 0.377)

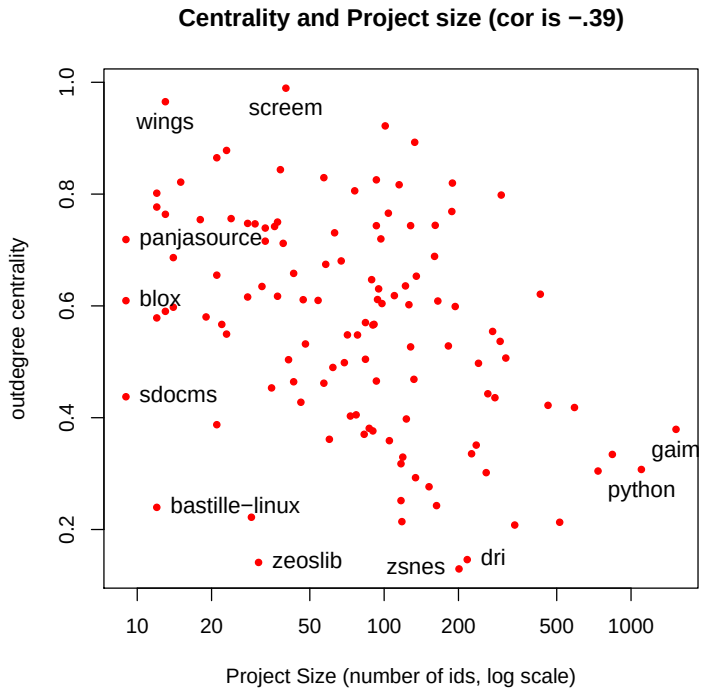

Figure 11: Plot of project centralization vs project size as measured by number of posters

correlation is significant with $r = -0.39$ ($N = 120$ and $p < 0.01$) and is illustrated in Figure 11.

A possible interpretation of this finding is 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 might be an aggregate of smaller projects, resulting in what might be described as a 'shallot-shaped' structure, with layers around multiple centres. Certainly Figure 11 indicates through its empty upper right corner that there are no highly centralized large projects.

This finding provides an alternative interpretation of our study which might yet reveal consistency across FLOSS projects by redefining the project as a group of sub-projects. However there are a number of confounds which would need to first be assessed before such basing such a finding on this correlation. Firstly the decentralization found in larger projects might be caused by a change of leadership, something which is more likely to happen in a large, and therefore probably older, project. And secondly, there would need to be an independent measure of modularity to provide the base-line to compare the 'centres' revealed through SNA. Such a baseline might be provided by assessment of modularity in the code-base.

Conclusions

Our study demonstrates that a particular pattern of communications centralization or decentralization is not a characteristic of FLOSS projects when engaged in the task of bug-fixing. This finding

is limited in what it tells us about social structure of FLOSS projects in general because it is based only on communications in the bug tracker system on SourceForge. However, as we have argued, the communications structure of a project is an important element in understanding a project's practices. Limiting enquiry to SourceForge actually strengthens our results because our results run against the expected bias; we found high range and a wide distribution, rather than tool biased similarity.

It is too early to make overly broad claims regarding the social structure of FLOSS projects but this study suggests that the received wisdom regarding the structure of communications in FLOSS projects does not agree with the empirical analysis. It should therefore not be taken for granted that FLOSS projects automatically inherit the practices and characteristics that have been found in case studies or in the projects on which the practicioner-advocates base their understanding of the FLOSS phenomenon. Further examination of other forms of social structure are required to round out and complement the findings of this paper. For example it would be useful to look at the communications in mailing lists, irc channels and, through interviews, face to face or telephone communications both independantly and as a whole. It would be very valuable to examine the development of these networks over time.

Our findings raise a number of important questions. Firstly, if the practitioner-advocates are correct in linking social structure to the advantages of FLOSS development, does the lack of consistency mean that not all FLOSS projects have access to these advantages? For researchers the question of where to draw the boundaries of the FLOSS phenomenon becomes more important: research on FLOSS practices that defines its subject by reference only to the license employed has, it seems to us, a heavy burden to demonstrate that it is studying similar projects and teams. For example, it might be necessary to treat sub-projects of large projects as basically separate projects, or it might be necessary to acknowledge that the practices of interest are shared by proprietary development teams employing, for example, agile software development techniques and thus expand the scope of the phenomenon beyond FLOSS.

The variance in communication structure potentially makes social structure even more interesting because it is way to differentiate between FLOSS projects. As an example of such usage we showed a negative relationship between project size and communication centralization. On further investigation this might be found to indicate the importance of achieving modularity in a project that is seeking to grow. It is not clear yet though if the projects that grow are the ones that have modularity or whether the inability to sustain centralized communications beyond a particular point is a driver of modularity. Certainly if growth is a goal of development, then the ability to achieve modularity could be a crucial success factor. Our analysis therefore calls for more detailed investigations of crucial phases in project growth and longitudinal analysis of FLOSS social structure.

About the authors

Kevin Crowston is an Associate Professor of Information Studies at the Syracuse University School of Information Studies. He is currently program director for the School's PhD in Information Transfer. Prior to moving to Syracuse, he taught for five years at the University of Michigan Business School.

He received his A.B. (1984) in Applied Mathematics (Computer Science) from Harvard University and a Ph.D. (1991) in Information Technologies from the Sloan School of Management, Massachusetts Institute of Technology.

His current research interests focus on new ways of organizing made possible by the use of information technology. This work approaches the issue in several ways: empirical studies of coordination-intensive processes in human organizations (especially virtual organization); theoretical characterizations of coordination problems and alternative methods for managing them; and design and empirical evaluation of systems to support people working together. For more information, please visit http://crowston.syr.edu .

James Howison is a doctoral student at the Syracuse University School of Information Studies. His research interests include the social science of software engineering and 'wireless grids' (distributed ad hoc resource sharing).

In 1998 he received his honors undergraduate degree in economics and politics from the University of Sydney. In 2001 he undertook graduate study in Software Engineering at the University of New South Wales before transferring to the Syracuse University School of Information Studies Ph.D. program in 2002.

He was recently published in IEEE Internet Computing and has presented at the International Conference on Information Systems (ICIS), the annual conference of the Association for Public Policy Analysis and Management (APPAM) and the International Conference on Software Engineering (ICSE). He is a sometime contributor to Bibdesk, an open source bibliographic manager for OS X, but certainly wouldn't claim to be central ;) Contact him at jhowison@syr.edu.

Acknowledgments

The Authors would like to thank Hala Annabi, Chengetai Masango, Joseph Davis, Anand Natarajan and Yeliz Eseryel for their assistance and comments on drafts. The Authors gratefully acknowledge the support of NSF Grants 03-41475 and 04?14468.

Notes

[1]http://incubator.apache.org

[2]http://floss.syr.edu

[3]The authors are making this paper available as a script that drives each step of the analysis and finally produces the paper in PDF form. This is done to encourage the FLOSS research community to both use and truly peer review our scripts and analysis. We hope that others will wish to build from this data-set and tools.

[4]The authors outline the difficulties, both theoretical and practical, of basing analyses of FLOSS purely on SourceForge in Howison and Crowston (2004). For this study however the possible selection bias can be understood: if there is a systemic effect of the SourceForge tools it should be to promote similarity in project structure. In our findings we review this possible impact.

[5]At least one project, `dri` now links to a `bugzilla` installation rather than the SourceForge tracker, but up until 2002 the tracker was the central repository. Because our analysis is cumulative it is not time sensitive and the data in the tracker, though older, is still a valid depiction of the communications structure at the time it was the central repository.

[6]Note that a 'circle' or 'ring' structure in which every person spoke only to those on their 'left' and/or 'right' would also produce a network with equal centrality scores and thus be a decentralized network. However this structure seems unlikely to be found within FLOSS development teams and we exclude it from discussion for reasons of clarity.

References

V Batagelj and A Mrvar, 1998. "Pajek: Program for large network analysis," *Connections*, volume 21, number 2, pp. 47–57.

Carter T Butts, "The SNA Package for R," , at http://erzuli.ss.uci.edu/R.stuff/, accessed 24 October 2004.

Alan Cox, 1998. "Cathedrals, Bazaars and the Town Council," *Slashdot*, (13 October 1998), at http://slashdot.org/features/98/10/13/1423253.shtml, accessed 24 October 2004.

Kevin Crowston, 1997. "A coordination theory approach to organizational process design," *Organization Science*, volume 8, number 2, pp. 157–175.

Cristina Gacek, Tony Lawrie, and Budi Arief, 2001. "The many meanings of open source," Technical Report 1, DIRC—Interdisciplinary Research Collaboration in Dependability, at http://www.dirc.org.uk/publications/techreports/papers/1.pdf, accessed 24 October 2004.

James Howison and Kevin Crowston, 2004. "The perils and pitfalls of mining SourceForge," in: "Proc. of Mining Software Repositories Workshop at the International Conference on Software Engineering (ICSE)," Edinburgh, Scotland, at http://floss.syr.edu/.

T Kamada and S Kawai, 1989. "An algorithmn for drawing general unidirected graphs," *Information Processing Letters*, volume 31, pp. 7–15.

Sandeep Krishnamurthy, 2002. "Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects," *First Monday*, volume 7, number 6 (June), at http://firstmonday.org/issues/issue7_6/krishnamurthy/index.html, accessed 24 October 2004.

Ko Kuwabara, 2000. "Linux: A Bazaar at the Edge of Chaos," *First Monday*, volume 5, number 3 (March), at http://firstmonday.org/issues/issue5_3/kuwabara/index.html, accessed 24 October 2004.

Audris Mockus, Roy T Fielding, and James D Herbsleb, 2002. "Two Case Studies Of Open Source Software Development: Apache And Mozilla," *ACM Transactions on Software Engineering and Methodology*, volume 11, number 3, pp. 309–346.

Jae Yun Moon and Lee Sproull, 2000. "Essence of Distributed Work: The Case of the Linux Kernel," *First Monday*, volume 5, number 11 (November), at http://firstmonday.org/issues/issue5_11/moon/index.html, accessed 24 October 2004.

NetMiner Webpage at http://www.netminer.com, accessed 28 April 2002.

R Development Core Team, 2004. *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, at http://www.R-project.org, accessed 24 October 2004.

Eric S. Raymond, 1998a. "The Cathedral and the Bazaar," *First Monday*, volume 3, number 3 (March), at http://www.firstmonday.org/issues/issue3_3/raymond/, accessed 24 October 2004.

Eric S. Raymond, 1998b. "Homesteading The Noo-Sphere," *First Monday*, volume 3, number 10 (October), at http://www.firstmonday.org/issues/issue3_10/raymond/index.html, accessed 24 October 2004.

Patrick Royston, 1982. "An Extension of Shapiro and Wilk's W Test for Normality to Large Samples," *Applied Statistics*, volume 31, pp. 115–124.

Walt Scacchi, 2002. "Software development practices in open source communities: A comparative case study," (Position Paper), at http://opensource.ucc.ie/icse2001/scacchi.pdf, accessed 2 February 2004.

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