

Studying Free Software with Free Software and Free methods
A paper for the Australian Open Source Development Conference
1–3 December 2004, Melbourne, Australia*
<http://osdc.com.au>

James Howison

November 8, 2004

Why research Free Libre and Open Source Software?

Free, Libre and Open source software (FLOSS¹) is an intriguing subject for study. Academic studies of FLOSS have quickly expanded over the past 5 years approaching the topic from software engineering, economics, information systems, sociology, social psychology to name just a few. The pre-print repository at MIT now has over 192 papers and even that is only a small cross-section of the research of FLOSS. There are a number of review papers, such as Rossi (2004), which provide a useful introduction to the scope of academic research on FLOSS.

It is to be hoped that this research is also relevant to FLOSS practitioners—not just to the Ivory towers of the academy. In our research we focus on practices rather than motivations and we hope that our research will help to satisfy natural curiosity of developers as well as help the community identify the most effective practices for developing FLOSS.

This paper briefly describes our recent research into the practices of FLOSS development teams using social network analysis and describes my efforts to do it with free software and free methods. It will probably come as no surprise to those familiar with academic research that the stated values of openness and collegiality are often not put into practice. Academics often jealously guard “their data” and under-describe “their methods” leading to publications which, while often impressive, can be, at the margins, more like patent disclosures than documents that others can learn and build from to test and improve knowledge.

Ultimately I hope that using free software and methods will lead to increased reproducibility, consistency and learning across academic studies of FLOSS and beyond. It is often said that FLOSS learnt from academia—it seems that free software methods can now teach in return.

*This work was supported by NSF Grants 03-41475 and 04-14468. See <http://floss.syr.edu> for more details and further work.

¹The acronym FLOSS has become accepted in academia as a term that includes both free and open source software without eliminating the distinction between them. Pragmatically, the author particularly likes it because the main alternative, F/OSS, can't be used in filenames!

Studying the social structure of FLOSS development

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. Dramatizing this perception, Eric Raymond coined his well-known comparison, “The Cathedral and the Bazaar” Raymond (1998) 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, 1998).

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.

This is important because, as we show in Crowston and Howison (2004), many of supposed strengths of FLOSS development are closely linked to the unique communications or social structure of the projects. 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.

Our research, therefore, questioned 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 we assessed the consistency of communications centrality in bug-fixing processes to see if there was consistency across FLOSS projects. The full analysis is reported in Crowston and Howison (2004) which is freely available online.

Gathering and Parsing Data

One of the great promises of research into FLOSS is that there is an abundance of data freely available as a by-product of the repositories that projects use to coordinate their activities, storing their communication and code. One of the oldest and best known repositories is SourceForge. In a perfect world access to the publicly available data would be available to academics (and others) as a data-base dump.

However it is not as simple as one would like to gain access to a database dump from Sourceforge (although they have provided dumps to many academics those are not typically able to be shared). Currently Sourceforge is working with a group of academics to make data relevant for research available regularly however until that time one is limited to data collection via spidering. This is not without its challenges and pitfalls which we reported in Howison and Crowston (2004).

In the course of our research we wrote a framework of scripts to collect data from Sourceforge. The foundation of these scripts was the excellent WWW::Mechanize which enabled programmatic navigation through the forms interface provided by Sourceforge to download the 61,068 bug reports across the sample of 120 projects that we required for our analysis.

Parsing the downloaded HTML to extract the data was a challenge ready-made for the Perl regex functions.

For social network analysis we only needed to parse the source and temporal order of the conversation made-up by the comments in the sourceforge bug tracker. Before groking the innate ability of regexs to handle this task I admit to spending an embarrassing amount of time attempting to use `HTML::Tokenizer` to access the data, somewhat guided by the admonitions in the O'Reilly texts that parsing html with Regex is to be avoided. In the end I was able to use these basic regex's to parse the data needed from the html (after slurping in the file to a single string).

```
# Do parsing, building up data in %item.
( $item->{'ID'}, $item->{'Title'},
  $item->{'SubRealName'}, $item->{'SubmittedBy'} ) = $fileContent =~
# Using {} for the delims rather than /
m{
    <H2>\[\s(\d+)\s\]\s(.*)</H2>
    .* #skip
    <TD><B>Submitted\sBy:</B><BR>(.*?)\s*\((.*?)\)</TD>
}xs;
# \s means that .* matches new-lines \x means that white space and comments are allowed
```

The follow-up comments were even more fun to parse. They are contained in reverse chronological order within a series of `<PRE>` tags and there can be an unlimited number of them. While the initial code using HTML parsing was tortured and unreliable, the regex code is one of the most satisfying lines of code I have written. It handled the challenge of an unknown number of repeated follow-ups like a champion:

```
# This find each of the textual follow-ups in order and puts them into an
# array The .*? is a non-greedy matcher (so it stops the pattern at the first
# <PRE> not the second!) and the /g finds all occurrences.
# See ch06_16 in the O'Reilly Perl Cookbook

my @followupsRaw = $fileContent =~ m{<PRE>(.*?)</PRE>}gs;
```

I was then able to pull the sender and time out of these strings.

Storing and sharing data for analysis

The task of storing the data for later analysis was trivial for the social network analysis there is an abundance of other data that needs to be stored for other analyses. We, together with another academic engaged in analyses on the Sourceforge data, Megan Conklin (Conlin, 2004), are in the process of developing a shared data-storage facility to provide researchers with access to a consistent, open data set of data collected from the full range of project repositories, whether it be through spidering or gaining access to quality database dumps.

An open comparable data storage facility is a vital to facilitate reproducible analyses which other academics can learn from and extend. We are calling the project `ossmole`² and host our development on sourceforge. We have developed a database which contains a record for each donation of data about a project and, since the provenance of data is vital to auditable research, stores the origin and transforms performed on each data-source. One challenge faced is ensuring that the data inserted is audited for consistency and validity. This is an open problem but we currently use bounds checking to ensure that entered data is within known bounds and is in reasonable agreement with other known good data.

²<http://ossmole.sf.net>

Sharing the analyses == open source research

One of the enduring frustrations I as an open source developer have faced as I become socialized to the academic community is an inability to “view source” on a research paper. I had become so accustomed to learning from others activities, something which I first experienced in html coding and later in FLOSS software proper. Academic research papers are designed to provide their results and sufficient detail to convince the reader of their validity—they are not really designed to teach others how to reproduce their analyses. This was, perhaps, legitimate in the days when the production of journal papers was expensive and space thus limited. Today with broad access to the Internet it is no longer an optimal situation.

Our social network analysis, of which brief results are presented below, was undertaken using the `sna` package which is part of the open source statistics package, `R` (R Development Core Team, 2004; Butts). Today there is a CPAN package available (`Statistics-R`) that links perl and the `R` executable but that was not available when we wrote our code. Instead we prepared a temporary file with `R` commands and ran the binary from within perl, reading the results from a file written to by `R`.

We have made the scripts for our analysis available through CVS in the `ossmole` project and intend to continue to do so. In fact we intend to release the paper, once it is accepted for publication, as a perl script. Yes Perl can actually write academic papers! I have a script that is able to drive the entire paper production process, from the download of data, to parsing, to analysis, to creation of graphics and, using `latex` and `bibtex`, the production of the actual PDF copy of the paper. Even if the knowledge in the paper is not that useful at least others will be able to “view source” and learn from the analysis, statistics and even layout tricks that lead to the full paper.

It is our hope that others will follow this lead allowing better reproducibility, more informed critique and community learning which will save significant time and advance the analyses towards usefulness more quickly.

So what did we find anyway?

The major findings from our social network analysis are depicted in the figures below. Full results are reported in Crowston and Howison (2004).

Figure 1 is a histogram of the project bug interaction centralization scores, calculated from dichotomized data, dropping ‘nobody’ interactions. Centralization scores are calculated by examining the inequality in individuals centrality scores. A perfectly decentralized network has a centralization score of 0.0 indicating perfect equality—each individual is as central as any other. A score of 1.0, on the other hand, indicates perfect inequality where one individual is central while all others are equally non-central.

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.

Figure 2 shows our finding that centralization scores are negatively correlated with the number of developers and active users who contributed to the bug reports. The correlation is significant with $r = -0.39$ ($N = 120$ and $p < 0.01$). This confirms the intuitive expectation that larger projects have more decentralized communications structures while smaller projects can be either centralized or decentralized depending, presumably, on the style of the project.

Figures 3 and 4 show graphically the more extreme examples of centralized and decentralized networks.

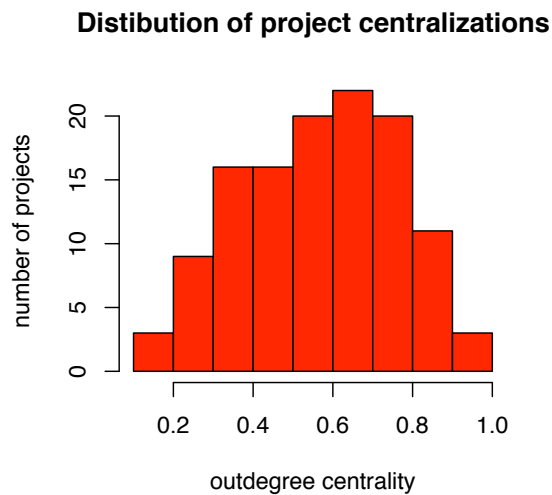


Figure 1: Histogram of communication centralization scores for projects engaged in the bug-fixing process.

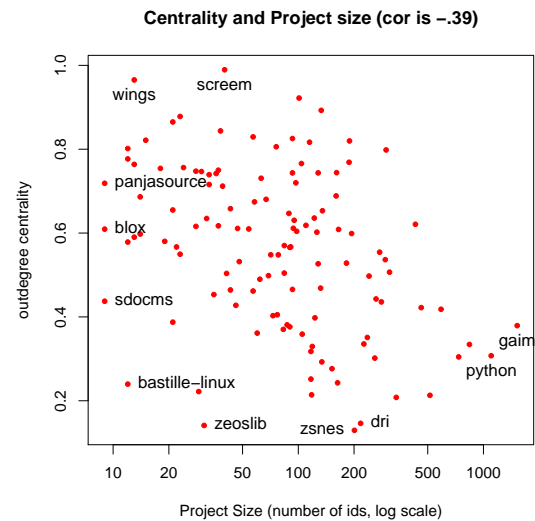


Figure 2: Plot of project centralization vs project size as measured by number of posters

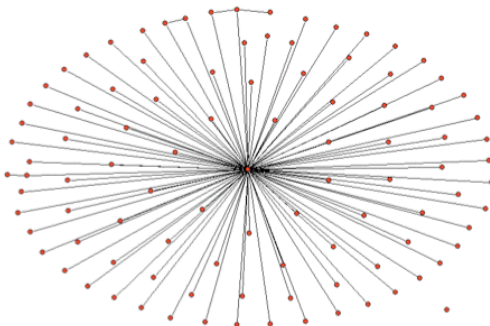


Figure 3: Plot of interactions for curl, a highly centralized project (centralization = 0.922)

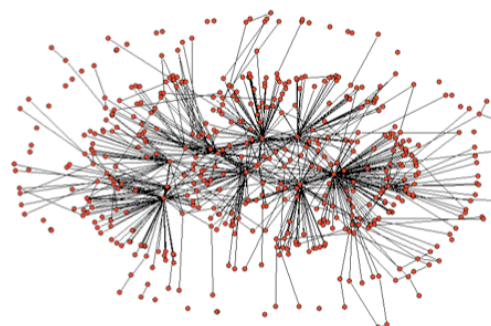


Figure 4: Plot of interactions for squirrelmail, a decentralized project (centralization = 0.377). Note the appearance of modularity in the network.

Figure 4 clearly shows a clustering of communications, some individuals in the periphery speak to particular individuals in the core and not to each other or other members of the core. This suggests that one explanation for the correlation depicted in Figure 2 is increasing modularity of communications.

Further work

We hope to test this hypothesis of modularity through a core-periphery analysis of the social networks and a comparison of that with modularity in the code-base. The accepted algorithms for discovering core-periphery relationships are not available in the R `sna` library so we intend to work with the authors to implement the needed techniques and contribute them to the project.

We are also conscious that our analysis has taken communications that occurred over time, in some cases up to three years, and ‘flattened’ it. We are exploring analyses which allow us to trace the development of the networks over time and hope to use these dynamic analyses to trace the entry of individuals to the network

and to describes their paths to (or from) core positions in the project teams.

Finally we are pursuing an analyses that examines how quickly projects are able to close their bugs in order to assess the efficacy of practices of bug prioritization and developer assignment.

We hope that our work will encourage others to share their data and analysis more widely—after all we are all studying Free and Open source software development, and it is only right that we learn from their successes.

References

- Carter T. Butts, “The SNA Package for R,” , at <http://erzuli.ss.uci.edu/R.stuff/>, accessed 24 October 2004. 4
- Megan Conlin, 2004. “Do the Rich Get Richer? The Impact of Power Laws on Open Source Development Projects,” in: “Proceedings of Open Source 2004 (OSCON),” at http://www.elon.edu/facstaff/mconklin/pubs/oscon_revised.pdf. 3
- 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. 2
- Kevin Crowston and James Howison, 2004. “The social structure of Open Source Software development teams,” *under review at First Monday*. 2, 4
- James Howison and Kevin Crowston, 2004. “The Perils and Pitfalls of Mining Sourceforge,” in: “Proc. of Workshop on Mining Software Repositories at the International Conference on Software Engineering ICSE,” at <http://floss.syr.edu>. 2
- 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. 4
- Eric S. Raymond, 1998. “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. 2
- Maria Alessandra Rossi, 2004. “Decoding the “Free/Open Source (F/OSS) Software Puzzle”: a survey of theoretical and empirical contributions,” *Quadreni*, volume 424 (April), at <http://opensource.mit.edu/papers/rossi.pdf>. 1