Community Structure in OSS Projects

Christian Bird*
Dept. of Computer Science, Kemper Hall,
University of California, Davis,
Davis, California Republic.
cabird@ucdavis.edu

Abstract

The structure and organization of people who work on any complex software project can in large part affect the quality and design. Unlike traditional software, Open Source Software (OSS) Projects have no mandated organizational structure. Rather the community is a self organizing and very dynamic entity. Due to the wealth of recorded information regarding open source projects, the organization of the community and changes to it can be observed and analyzed. We believe that subcommunities form within large and complex OSS projects such as the Apache[1] webserver due to the number of volunteers and the various subsystems within the codebase. Evaluating whether these subcommunities exist and examining their behavior can help us better understand how self organizing software communities work. In addition, this information may shed light on how OSS projects are able to produce high quality, complex software.

We have mined the activities in the Apache development mailing list and software repository. From this data we have been able to reconstruct the social network of volunteers who have contributed to the Apache project. Using recent advances in complex network theory, we have extracted and evaluated subgroups from within the social network and observed their changes over time. Our results show that there are indeed a number of subgroups within the Apache social network and that there is a correlation between the age of the project and "strength" of these subcommunities. We present our findings in this paper.

1 Introduction

Open Source Software (OSS) projects differ from traditional software development efforts in a number of ways. Foremost among these differences are the facts that those who contribute to the project are not directly compensated financially (though there are cases where non-associated companies or organizations pay employees to contribute to OSS projects) and that no one is forced to work on a particular portion of the project. In a traditional business setting, the organization of a software development team is designed and mandated by management within a company. Members of the team must follow direction or risk losing employment. In the OSS world, there is no program manager to impose organizational restrictions on contributors.

*Some of the data gathering and processing work was performed prior to this course for other research. We would therefore like to acknowledge the help of Prem Devanbu and Alex Gourley.

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

Even if there were, no financial penalty would be associated with not following the restrictions. Although each OSS project is led and managed by an individual or core group, their role is more focused on managing the software artifact itself rather than the contributors. Contributors may come and go as they wish and may choose to help the project in any number of ways by submitting bug reports, lending technical knowledge, writing documentation, improving the source code in various ares of the code base, etc. Thus the community of contributors for OSS projects is for the most part self-organizing.

It is striking, therefore, that although this bazaar style of management differs dramatically from the organized, hierarchical, Cathedral style of traditional software efforts, the complexity and quality of the software produced by some OSS projects is on par with, and in some cases, arguably superior to, the results of the former [17, 11]. The fact that this self-organizing style yields fruit makes it an important area of study. We feel that an understanding of the OSS methodology and what makes it successful can be an invaluable resource in the software engineering field. Furthermore, OSS projects have certain characteristics that lend themselves to the analysis of their organization and process.

The community on the Apache developer mailing list is concerned primarily with the software itself. The webserver is a rather large and complicated artifact. Few, if any, developers are experts in all areas of it's code. In addition to discussions regarding source code, the mailing list also contains communication regarding topics such as the build system, documentation, and high level architecture. Similar to groups of development teams in a software company, we believe that within the community of developer mailing list participants, there are self-organizing subgroups of people that form as volunteers migrate towards particular subsystems or areas of work. We can use archives of the developer mailing list to determine which individuals were in communication with each other to create a social network of the participants. A key property of a social network is it's *community structure*, the appearance of densely connected groups of nodes with only sparse connections between groups[16]. Our goal is to extract the community structure, or subcommunities, from this social network and observe the changes in it's structure over the life of the project. In addition, we would like to examine the relationship between these subgroups and the software artifact itself.

We are hoping to answer the following questions:

- *Do subgroups form within the developer community and can we extract this structure using existing methods in network theory?*
- *How does the social network and the community structure within it change over time?*
- *Is there a structural difference between developers and non-developers in the community?*
- *What relationship, if any, exists between the social structure and the software artifact itself?*

In section 2 we discuss other research that has examined OSS communities and include the differences and similarities to our work. In addition, we include a brief survey of work related to community structure theory and it's applications. Our methodology and results are presented in section 3. We discuss these results, examine the strengths and weaknesses of our approach, and give further areas of study in section 4.

2 Related Work

Our work combines social networks in OSS communities with community structure theory. We present related work in both areas below.

2.1 OSS Social Analysis

There is a significant body or research devoted to examining OSS communities. Here we survey some of the more recent work involving social processes and social networks in that context.

Nicolas Ducheneaut[5], examined the python development community and presented a case study of the process by which an individual evolved from a casual mailing list peruser to a full-fledged core developer who gained access to the software repository and added his own module to the python standard library. His was a qualitative examination of the community of one open source project, but his proposed explanations of how the community works and makes decisions has been a key source of motivation for this and other works of ours[1, 2].

Social networks among developers have been studied from perspectives differing from ours. Xu *et al* [20] consider two developers socially related if they participate in the same project. This is a macroscopic view of a large portion of the entire OSS community. Our view is to consider contributors related if there is evidence of email communication; this is arguably a more direct evidence of a social link. In addition, because we examine each participant on the mailing list, we are able to capture a view beyond just the actual project developers.

Wagstrom, Herbsleb and Carley [18] gathered empirical social network data from several sources, including blogs, email lists and networking web sites, and built models of their social behavior on the network; these were then used to construct a simulation model of how users joined and left projects. Our goal is empirical rather than to run a simulation; we explicitly wish to study the structure of the community within an individual project.

Crowston & Howison [4] use co-occurrence of developers on bug reports as indicators of a social link. They empirically demonstrate that the social networks of smaller projects are more central than those of larger projects. Presumably larger projects decentralize, to simplify Communication & Coordination activities. This observation is one of the key motivators behind this work as we hypothesize that as a project grows and becomes decentralized, subcommunities form naturally.

Commit behavior in versioned repositories has been used as an indicator of social linkage. Lopez-Fernandez *et al* [12] consider two developers to be linked if they commited to the same module. The resulting social networks are similar in structure to ours. The work of De Souza *et al* [3] is similar, except that they study files instead of modules. Developers become more "central" in the social network over time. They found that code ownership in some parts of the system was more stable than in others. Finally, we note that these papers study collaboration networks, whereas our focus is more on communication networks. The relationship between the two is a subject of our current research.

In previous work[1, 2] we examined social networks created from mailing list archives and looked at the differences between developers and non-developers from a social network metrics standpoint. We also examined the correlation between development activity and social network status of developers. In this work our goal is to extract the subcommunity structure from the same social networks and examine how it changes over time.

2.2 Extracting Community Structure

In the past few years, there has been an emergence of complex network theory in the areas of physics and mathematics. We are specifically interested in community structure theory, pioneered by Mark Newman and Michelle Girvan. We use methods from community structure theory to find community structure within OSS social networks, but researchers have already applied these methods in other domains. In this section we present recent work related to community structure theory and it's applications.

In 2002, Newman and Girvan[7] presented their method of determining community structure based on edge betweenness, a social network measure created by Linton Freeman[6, 19]. They evaluated their method on a number of networks with divisions known a priori such as computer generated networks, NCAA football teams (divided by conference), and the somewhat famous (in SNA lore) Zachary "karate club"[21] with fairly accurate results. They also gave examples of results on networks for which the community structure was not known beforehand, such as the collaboration network of scientists at the Santa Fe Institute and the food web in Chesapeake Bay.

Since then, a number of researchers have used this and similar methods to find community structure in existing networks. Guimera *et al*-[9] mined email logs from a company to create a social network and used the above method to extract the community structure from it. They discussed the results as the *informal networks* behind the formal chart of an organization and it's benefit as a management tool. Gleiser and Danon[8] created social networks tying jazz musicians together if they played in the same band and bands together if they shared a common musician. Their analysis revealed correlations between recording locations, racial segregation, and community structure. Zhi-Qiang Jiang *et al*[10] examined the network of equipment(e.g. towers, pumps, heat exchangers) and pipes to create a flow network in an ammonia plant in China. They found that it was indeed a small-world network with high modularity indicating strong community structure.

Newman presented a method for both finding and evaluating community structure for large networks based on ideas related to spectral partitioning just a few months ago[15]. Modularity, His measurement of the quality of the network partitions, allows different partitions of the same network to be compared. For large networks, the proposed eigen analysis method of finding community structure is shown to yield better results than previously published methods. In addition, the problem of finding the optimal community structure is shown (informally) to be NP-hard. Extensions to this work were published less than a month ago[2] which may have beneficial implications with regard to social network metrics[14]. The work of this paper is based partly on this method of finding community structure within networks.

3 Results

There were a number of phases involved in implementing and carrying out our experiment. We first mined the archives of the Apache developer mailing list[3]. Next, we created a social network of the participants in intervals of three months. The community structure of each social network was then determined and evaluated. Finally, we examined the changes in the social network and the community structure found within in it over time. The

[2]Some would say I'm a Mark Newman stalker. I say stand on the shoulders of giants...

[3]It is important to note that this step and portions of the social network creation were performed prior to this course

following subsections contains the details of each phase.

3.1 Mining the Mailing List

The first step in our search for community structure involved picking an OSS community and actually mining the mailing list. We decided on the Apache developer mailing list for a number of reasons. The Apache webserver is one of the best known pieces of open source software next to the Linux kernel. It is both mature and widely used (a recent survey showed that 68% of websites are powered by Apache[4]) and has a long history (over ten years). In addition, the size of the development community and the volume of the mailing list are large enough that we felt optimistic we would find strong community structure in it.

Since the developer mailing list is archived and publically available, downloading the raw email text was not difficult. We designed a database schema to hold the relevant information and wrote python scripts to parse the relevant information out of the archives. For this exercise, the pieces of data that we needed for each message were the date it was sent, the name and email address of the sender, the *message-id* header, and the *in-reply-to* header. The last two are particularly important as they allow us to reconstruct threads of conversation. Every message sent on the mailing list has a unique *message-id* associated with it. If the the *message-id* associated with message A appears in the *in-reply-to* header of message B, then B was sent in response to A. We feel that this is an indication of a social link between the sender of A and the sender of B. From the inception of the developer mailing list to August of 2005, 102,611 messages were sent. We were able parse 101,637 of these. Most parse failures were due to malformed headers in the email messages. The distribution of these messages by month is shown in figure 1.

[4]see http://news.netcraft.com/archives/2006/02/02/february_2006_web_server_survey.html

3.2 Creating the Social Networks

We can create links between senders of emails by looking at who responded to whom on the mailing list. By examining the messages sent over a period of time, we can thus create a network of participants in the Apache community. Unfortunately, one last hurdle remains before we can do this accurately. A number of participants on the mailing list used a multitude of email addresses. In order for our results to be valid, we needed to be able to attribute all messages sent by a particular person to one node in the social network, even if that person used more than one address. We call this problem email aliasing. In response to this problem, we have come up with a method that removes this aliasing and is able to fairly accurately determine which email addresses are attributable to the same person. Our method is based on fuzzy string similarity, clustering, heuristics, and manual post-processing. After applying our method, we found that there were 2544 unique participants on the mailing list over the time studied.

Once email aliasing is removed, creating the social network is simply a matter of running an SQL query on our database to get a list of who responded to whom (and how many times) and use the results to create an adjacency matrix. Because we are looking for community structure based on the social links between people, we do not include people whose messages did not receive responses and were not responses themselves. We have built a tool that can create a valued adjacency matrix corresponding to the social network on the mailing list for any time period. We decided to use three month intervals. This is based on a recent discus-

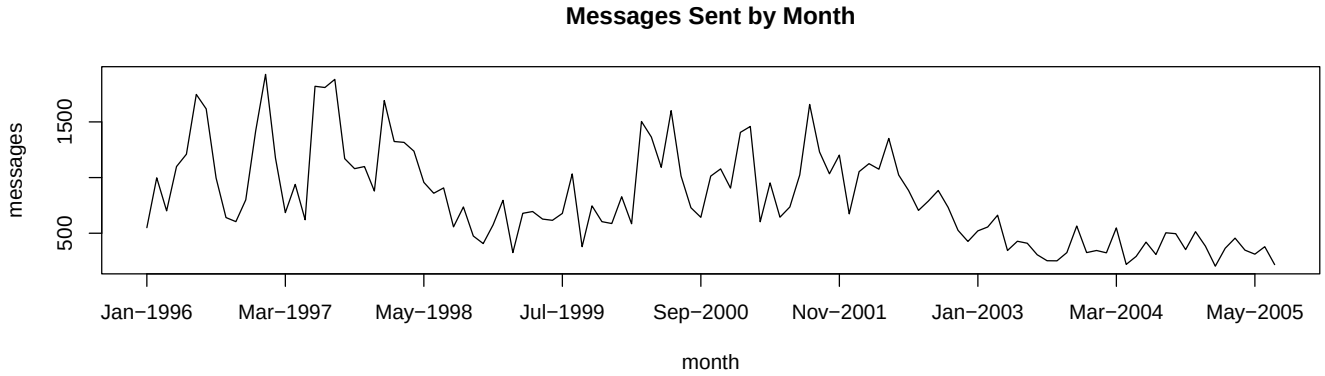

Figure 1: Number of messages sent to the Apache developer mailing list per month

sion at the workshop on Mining Software Repositories (MSR06) with other researchers who had found evidence that the project "memory" in OSS is somewhere around three months. The sizes of each of these networks is shown in figure 2.

3.3 Finding Community Structure

Once the social networks have been created, we can use methods from complex network theory to find and evaluate community structure in them. We use the method presented by Mark Newman in [15]. A brief overview of this algorithm is given below.

3.3.1 Community Structure Algorithm

In looking for community structure, our goal is a partition of the network into groups of nodes where the connections within the groups are dense and the connections between the groups are sparse. Newman and Girvan defined a measure, *modularity*, which quanitifies how strong the community structure is based on the denseness and sparsity of the intra and interconnections of the groups[16]. Suppose that we have a partition of a network into k communities. Let us define a $k \times k$ symmetric matrix e whose element e_{ij} is the fraction of all edges in the network that link vertices in group i to vertices in group j. Let us also define the row sums $a_i = \sum_j e_{ij}$. The modularity measure is then defined by

$$Q = \sum_i (e_{ii} - a_i^2)$$

Essentially, this measures the fraction of the edges in the network that connect vertices within the same group minus the expected value of the same quantity in a network with the same community divisions, but random connections between the vertices. Values for Q range from 0 (if the number of intra-community edges is no better than random) to 1. Modularity measures above 0.3 indicate strong community (or modular) structure. In partitioning the social networks, we want to find the partition that yields the highest modularity for the network.

It turns out that finding the partition that gives the highest modularity is most likely and NP-hard problem[15]. We therefore use an approximation method that appears to perform well. It works in the following way; Let A be a binary adjacency matrix for an undirected network with m edges and let k_i indicate the degree of vertex i. We can then create a matrix B, which we call the modularity matrix where

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

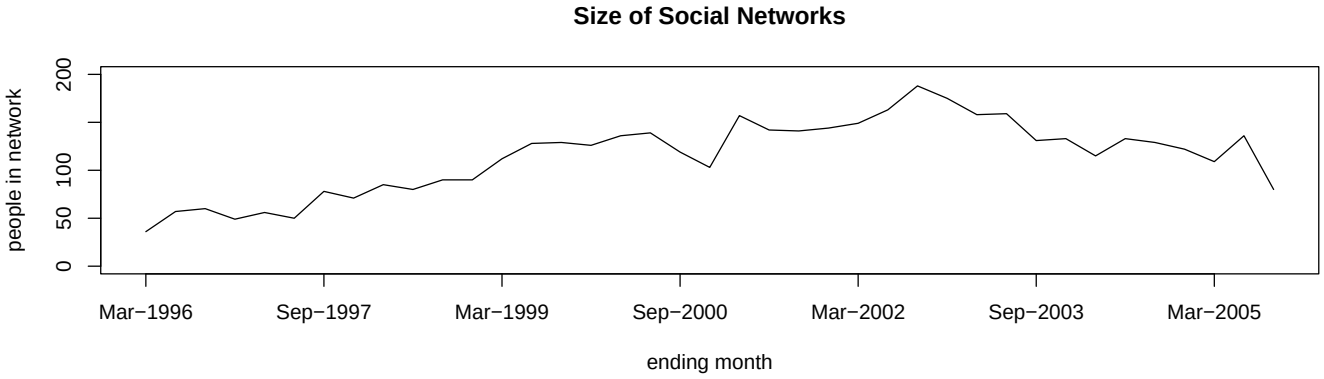

Figure 2: Size of each social network (each network created from data over a three month interval).

Each entry in B is the number of edges between nodes i and j minus the expected number of edges between them given their respective degrees if the edges are placed at random. The eigenvalues and eigenvectors of B are calculated and the eigenvector associated with the largest eigenvalue is examined. The nodes are then divided into two groups by looking at the sign of the each value in the eigenvector. Nodes with a positive value in the eigenvector are placed into one group and the rest are placed into the other group. The modularity of the partition is then recorded. Next we find, among the nodes, the one that when moved to the other group, yields the greatest increase in modularity. This process of moving individual nodes is repeated until each of the nodes has been moved. Then we search over the set of intermediate states occupied by the network to find the state with the highest modularity. Once we have our final two groups, the entire algorithm is applied recursively to each until no division of a group results in higher modularity[5].

3.3.2 Community Structure in Apache

The results of applying this algorithm to our Apache data are actually somewhat intuitive. During the early life of the project (1996-2000) the strength of community structure is relatively low. As the project matures, the community grows and the code becomes larger and more complex. The modularity also increases with age of the project and structure in the social network begins to emerge. A graph of the modularity of the community structure can be seen in figure 3 and the sizes of the partitioned groups for each time interval is in table 1. It's interesting to note that the strength of the community structure seems to be relatively independent of the number of emails sent on the mailing list for a given time period.

[5]For a more detailed explanation, including why this actually works, please see the original paper[15].

3.4 Examining Changes

In the last stage of our experiment, we want to look at how the community structure changes over time. More specifically, from one network to the next we'd like to know both the turnover (how many people left and how many joined the mailing list) and what fraction of participants change in the subcommunities found.

3.4.1 Community Turnover

To measure the turnover rate between the networks for two time intervals, we look at the number of people who left the mailing list and the number

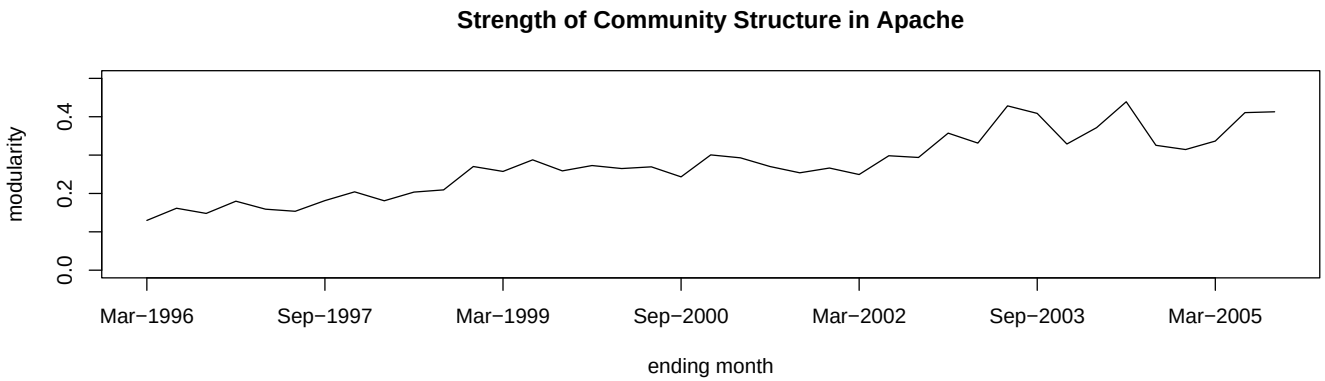

Figure 3: Modularity for the social networks over time on the Apache developer mailing list. Values at or above 0.3 indicate strong community structure.

interval	size	Q	group sizes	interval	size	Q	group sizes
Mar-1996	36	0.13	11:10:10:9	Mar-2001	157	0.29	24:21:20:19:17:16:15:14:12:2
Jun-1996	57	0.16	18:15:14:9:9	Jun-2001	142	0.27	35:21:21:15:13:11:10:5:3:3
Sep-1996	60	0.15	15:13:9:8:6:4	Sep-2001	141	0.25	37:23:20:20:17:14:13:4
Dec-1996	49	0.18	17:17:11:11	Dec-2001	144	0.27	32:25:23:21:17:15:11
Mar-1997	56	0.16	15:15:10:10	Mar-2002	149	0.25	50:28:21:18:16:13
Jun-1997	50	0.15	16:15:14:10	Jun-2002	163	0.30	37:31:27:23:22:21:8
Sep-1997	78	0.18	25:21:12:10:8:2	Sep-2002	188	0.29	31:28:24:19:18:17:13:11:8:7:6:5:3
Dec-1997	71	0.20	19:16:14:8:8:7	Dec-2002	175	0.36	26:25:21:19:18:15:13:11:9:4:4
Mar-1998	85	0.18	30:19:14:11:10	Mar-2003	158	0.33	42:42:37:18:12:10:4
Jun-1998	80	0.20	20:18:14:11:9:7:4:2	Jun-2003	159	0.43	37:23:22:17:16:13:2:2:2:2:2:1
Sep-1998	90	0.21	23:19:18:15:10	Sep-2003	131	0.41	16:15:15:15:14:13:8:7:4:2:2:2:1
Dec-1998	90	0.27	22:20:20:12:11:8:3	Dec-2003	133	0.33	22:22:20:16:16:13:10:7:4:2
Mar-1999	112	0.26	33:24:18:17:11:10:4:1	Mar-2004	115	0.37	26:23:19:15:14:9:9:8:7:1:1
Jun-1999	128	0.29	38:27:17:14:13:12:9:3	Jun-2004	133	0.44	25:22:20:15:15:15:4:3:2:2:2:2:1:1:1
Sep-1999	129	0.26	24:22:19:15:15:14:12:4:2:1	Sep-2004	129	0.33	30:22:18:17:14:3:3:2:2:1:1
Dec-1999	126	0.27	30:23:22:19:18:10:2	Dec-2004	122	0.31	21:20:18:14:12:11:11:9:6:1
Mar-2000	136	0.26	34:30:24:18:17:15:5:2	Mar-2005	109	0.34	28:23:22:18:14:10
Jun-2000	139	0.27	22:21:21:18:16:14:13:9:5:1	Jun-2005	136	0.41	26:18:17:16:15:14:9:6:3:2:2:2:2
Sep-2000	119	0.24	30:19:13:11:10:10:8:1	Jul-2005	80	0.41	19:13:6:4:2
Dec-2000	103	0.30	29:26:26:18:7:4				

Table 1: Information regarding the community structure of the social networks in Apache. (Q = modularity, size is the size of the entire social network for that time period)

of people who joined the mailing list. We have attempted to come up with a metric to quantify this phenomenon. Although we are sure that it can be improved, we feel that the following method does a reasonable job. We are just measuring the number of people who leave plus the number of people who join, divided by the sizes of the two networks. Let N_t be the set of participants in the social network at time time t. The network change from time t to $t+1$ is calculated as follows

$$C_{t,t+1} = \frac{|N_t \setminus N_{t+1}| + |N_{t+1} \setminus N_t|}{|N_t| + |N_{t+1}|}$$

To possibly give more intuition to this measure, this simplifies to

$$C_{t,t+1} = 1 - \frac{2|N_t \cap N_{t+1}|}{|N_t| + |N_{t+1}|}$$

where $I_{t,t+1}$ is the number of participants that are present in both networks (i.e. the size of the intersection of network actors).

A value of 0 means that no participants left or joined while a value of 1 indicates that no participants in the first network stayed. For networks of similar size, a value of 0.5 indicates that roughly half of the participants from the first network remained in the second as well. The number of people who stayed on the mailing list across time intervals, which we call *participant commonality*, and this rate of change are both shown in figure 4.

This graph shows that the turnover rate slowly grows from 0.3 in early 1996 to 0.6 in 2005. This may be a misleading result. When combined with the growth in participants on the mailing list, we can see that the number of people that stay on the mailing list for more than one interval is actually increasing in some cases, even though the turnover rate also increases. Towards the end of the period of time studied there appears to be an inverse relationship between the number of participants who stay on the mailing list and the turnover[6].

3.4.2 Subgroup Commonality

From the results of the partitioning step, we can see that the modularity, or strength of community structure, appears to increase over the life of the project. This indicates not only that subcommunities form, but that the intra-group ties become stronger while the inter-group ties become weaker. One question that we had about these communities was how dynamic they are. Put another way, do the participants in subgroups change much over time, or do groups of people in a group in one time interval tend to appear together in a subgroup in the following time interval. Our way of measuring this is by looking at the number of people who are in the same subgroup as a participant at one time period and observing what fraction of those people are in the same subgroup as the participant in the next time period. We call this *subgroup commonality*. Formally, let N_t be the set of participants in the network at time t and let $N_{t,p}$ be the set of participants in the same subgroup as participant p at time t. The subgroup commonality at time t is calculated as

$$C = \sum_{p \in N_t} \frac{|N_{t,p} \cap N_{t+1,p}|}{|N_t|}$$

The result of this is quite easy to interpret. If the subgroup commonality is 0.25 then the expected value of the number of people in the same subgroup as a particular person in the second network is one quarter of the people who were in the same subgroup as that person in the first network. If the subgroups don't change at all then this number will be 1. If all of the individuals in each subgroup split up then the value will be 0. After running this metric on our Apache data, the values tend to range from 0.2 to 0.4. This may sound low, but if 40% of the people in a participant's subcommunity in one time period are still in the same participant's subcommunity in the next, when there are anywhere from four to ten subgroups, that actually indicates a strong tendency to stay together. A question for further investigation is if people who "leave" a participant, migrate to multiple subgroups or stay together in one. Situations that are problematic

[6]This is not too surprising since, intuitively, they are opposites to some degree

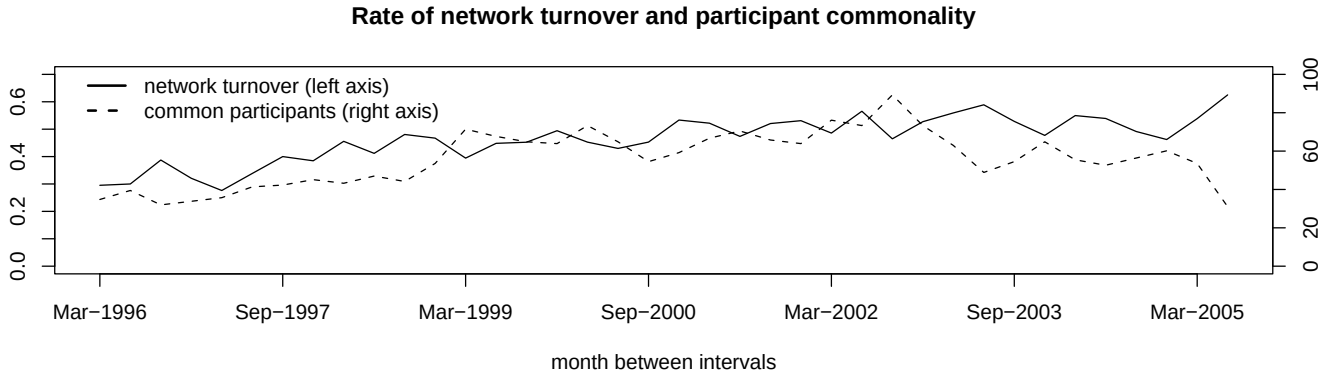

Figure 4: The solid line shows the network turnover rate (values on the left axis). The dashed line indicates the number of participants who stayed on the mailing list across time intervals (values on the right axis).

for this metric are those in which a subcommunity splits across a time interval or a group of people, who in reality form a loose subcommunity, that the algorithm splits into two. In either case, the average number of subgroup-mates common to participants in the first network drops to below 50% even though the subcommunity makeup hasn't changed dramatically. We hope to address this issue in the future. The results of running this metric on our Apache data can be found in figure 5.

4 Discussion

Our analysis has shown that community structure exists in the Apache developer mailing list and that across time intervals roughly 20% to 40% of the members of any particular subgroup stay together. In addition, we found that the strength of the community structure appears to grow over time and is relatively independent of the number of messages sent over a given time period.

As far as we can tell, this is the first analysis of community structure in an OSS project. That in and of itself is a strength of this research. We feel confident in the quality of the social networks that were produced and that they are relatively free of noise. Due to the modularity levels of the community structure found, especially later in the project's life, we feel that the partitions are representative of a real phenomenon and not artificially created.

There are some areas of our research that could be improved. The largest problem with our analysis is that our social networks are not valued. Because the partitioning algorithm is based only on the presence or absence of a social link, it does not take into account the strength of that link. We have the ability to quantify the strength of a link based on the number of emails exchanged between mailing list participants, but right now we're essentially throwing that information away. Modifying the algorithm to take this additional information into account would most likely produce superior network partitions (from a modularity standpoint). Many algorithms and analyses deal with binary networks. In [13] a method of interpretting weighted graphs as binary multigraphs is presented and some algorithms have been adapted in this way. We hope to use this approach to alter the algorithm in the future.

In addition, our metrics for measuring network turnover and subgroup commonality have not been thoroughly evaluated and may have shortcomings. We feel that they give the reader some sense of the

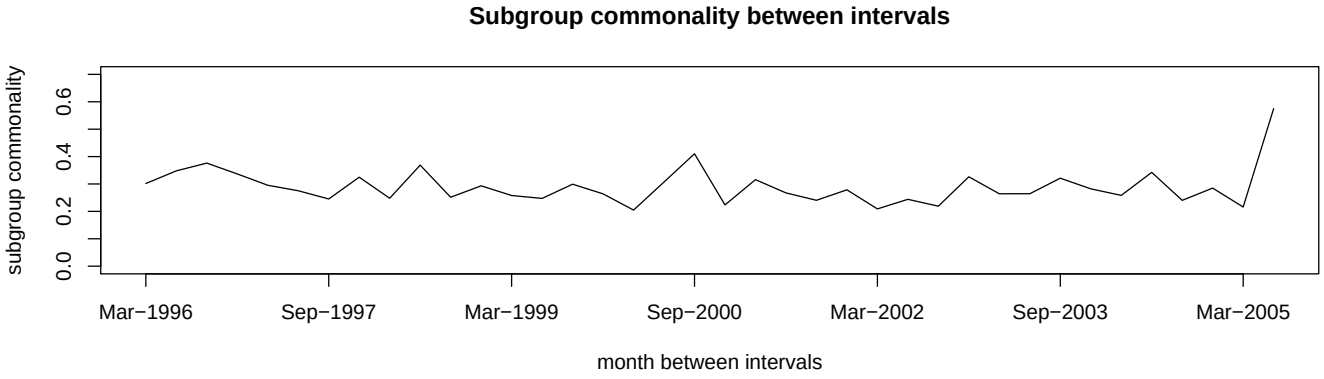

Figure 5: The group commonality for the subgroups in each social network for the Apache developer mailing list.

property of the network that we're trying to measure, but improvements can definitely be made. We would like to artificially create networks of varying sizes, degree distributions, etc. that have both high and low levels of the properties that we're trying to measure and evaluate multiple metrics to find which is optimal.

Although we have found that community structure exists in the Apache community, this work offers more questions than answers. What roles do developers and non-developers play in the subcommunities? Visually, what do the subcommunity networks look like? They rarely contain over 30 nodes, so patterns should be visible if they exist. Is there a relationship between length of time on a mailing list and the types of people that a participant is in a subcommunity with? Is there any relationship between the developers in a particular subcommunity and the code that they own or contribute to (i.e. can we validate Conway's Law in an OSS context)? Perhaps more importantly, are the results of our analysis unique to the Apache community, or representative of a more general OSS social process? We hope to perform the same analysis on a number of OSS projects in various stages of maturity.

Although this work is admittedly somewhat preliminary, we feel optimistic from the initial results that the answers to the above questions will yield insight into the workings of a successful OSS community. We plan to continue this work by addressing the shortcomings mentioned and investigating these and related questions during the coming summer.

References

[1] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 3rd International Workshop on Mining Software Repositories*, 2006.

[2] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks in postgres. In *Proceedings of the 3rd International Workshop on Mining Software Repositories*, 2006.

[3] J. F. P. D. Cleidson de Souza. Seeking the source: Software source code as a social and technical artifact, 2005. http://opensource.mit.edu/papers/desouza.pdf.

[4] K. Crowston and J. Howison. The social structure of free and open source software development. `opensource.mit.edu/papers/crowstonhowison.pdf`, November 2004.

[5] N. Ducheneaut. Socialization in an open source software community: A socio-technical analy-

sis. *Computer Supported Cooperative Work*, 14(4):323–368, 2005.

[6] L. Freeman. A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–41, 1977.

[7] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PROC.NATL.ACAD.SCI.USA*, 99:7821, 2002.

[8] P. Gleiser and L. Danon. Community structure in jazz. *Advances in Complex Systems*, 6:565, 2003.

[9] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in organisations. *Physical Review E*, 68:065103, 2003.

[10] Z.-Q. Jiang, W.-X. Zhou, B. Xu, and W.-K. Yuan. A complex ammonia plant network in chemical engineering, 2005.

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

[12] L. Lopez, J. M. Gonzalez-Barahona, and G. Robles. Applying social network analysis to the information in cvs repositories. In *Proceedings of the International Workshop on Mining Software Repositories*, 2004.

[13] M. E. J. Newman. Analysis of weighted networks. *Physical Review E*, 70:056131, 2004.

[14] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices, 2006.

[15] M. E. J. Newman. Modularity and community structure in networks, 2006.

[16] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.

[17] E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly and Associates, Sebastopol, California, 1999.

[18] P. A. Wagstrom, J. D. Herbsleb, and K. Carley. A social network approach to free/open source software simulation. In *Proceedings First International Conference on Open Source Systems*, pages 16–23, 2005.

[19] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge University Press, 1994.

[20] J. Xu, Y. Gao, S. Christley, and G. Madey. A topological analysis of the open source software development community. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 7*, 2005.

[21] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.