Chapels in the Bazaar? Latent Social Structure in OSS

Christian Bird, David Pattison, Raissa D'Souza,
Vladimir Filkov and Premkumar Devanbu
Dept. of Computer Science, Kemper Hall,
University of California, Davis,
Davis, California Republic.
cabird,dspattison,rmdsouza,vfilkov,ptdevanbu@ucdavis.edu

ABSTRACT

Commercial software project managers design project organizational structure carefully, mindful of available skills, division of labour, geographical boundaries, etc. These organizational "cathedrals" are to be contrasted with the "bazaar-like" nature of Open Source Software (OSS) Projects, which have no pre-designed organizational structure. Any structure that exists is dynamic, self-organizing, latent, and usually not explicitly stated. However, in large, complex, successful, OSS projects, we expect that sub-communities will form organically within the "bazaar" of developer teams. Studying these sub-communities, and their behavior can shed light on how successful OSS projects self-organize. This phenomenon could even hold important lessons for how commercial software teams might be organized. Building on well-established techniques for detecting *community structure* in complex networks, we extract and evaluate latent subcommunities from the email social network of several projects: Apache HTTPD, Python, PostgresSQL, Perl, and Apache ANT. We then validate them with software development activity history. Our results show that subcommunities do indeed form within these projects. We find, in other words, that "chapels" (if not cathedrals) spontaneously arise within the bazaar as OSS systems and the teams evolve. We also find that these subgroups manifest most strongly in technical discussions, and are significantly connected with collaboration behaviour.

1. INTRODUCTION

Brooks, in his seminal work *The Mythical Man-Month* [11], noted the scaling issues that arise in large software teams: the potential number of communication channels increases as the square of the number of people working on a project, thus quadrupling when the team size is doubled. Clearly, without organization of some kind, both within the software and the community that develops it, there is a limit to how much projects can be scaled.

This work was supported by a grant from the National Science Foundation Grant no. NSF-SoD-0613949 and software donations from SciTools and GrammaTech Corporations.

FSE '08 Atlanta, USA

In traditional, commercial software projects, the response to the Brooksian critique of large teams is to divide and conquer, *by fiat*. The system is deliberately divided into smaller components and the developer pool is grouped into manageable teams which are assigned to the components. With well-defined interfaces, the teams' efforts are confined to smaller groups, and the co-ordination needs are moderated. Software design principles such as separation of concerns [52] play a part in this, as does "Conway's Law" [15], which connects artifact structure with organizational structure.

By contrast, Open Source Software (OSS) projects are not formally organized, and have no pre-assigned command and control structure. No one is forced to work on a particular portion of the project. Team members contribute as they wish 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. Raymond [53] contrasts the *cathedral* model of the traditional software project with the *bazaar* model of the open-source project. The question then arises, is the social structure of OSS projects actually unorganized and free-for-all, and "bazaar-like", in contrast to the structured, hierarchical, "cathedral" style of traditional commercial software efforts? Or does the Bazaar have some structure of its own? Are there latent, dynamic, self-organizaing "chapels" that rise and fall within the bazaar? Are these "chapels" significantly related to technical focus? Are some discussions more fragmented into sub-communities, and others more free-for-all and bazaar-like?

In this paper, we discuss our empirical study of the latent social structure of open-source projects, and discuss just these issues. In the next section, we discuss the background, and state our research questions.

2. BACKGROUND

Despite this perceived lack of mandated organization, there are OSS projects with large developer pools that produce software of complexity and quality that rivals their commercial counterparts [53, 36]. How do these projects cope with the organizational hurdles that confront all large engineering efforts ?

We have empirically studied the social organization of the community of participants on the developer mailing lists for the Apache Webserver (hereafter referred to as Apache), Apache Ant (referred to as Ant), Python, Perl, and PostgreSQL projects. Each of these projects is mature and stable, has a large and complex code base comprised of multiple subsystems, and has a recorded history of many years. They

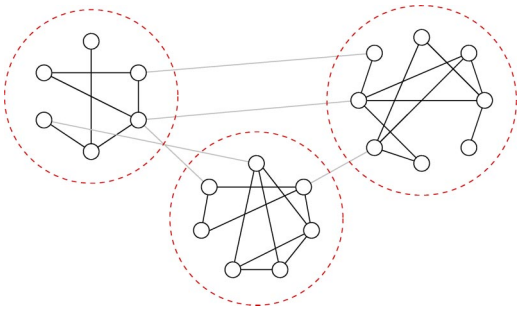

Figure 1: A network with strong community structure.

all also have sizeable teams, ranging in size from 25 developers to nearly 100[1]. There are of course much larger numbers of participants on the developer mailing lists [21], sometimes numbering in the thousands. The developer mailing lists are highly task focused; by community norms, all substantive discussions related to the system and development tasks occur on these lists. All correspondence is archived. The source code authorship history is also available from versioned source code repositories. The size and extensively archived history of these projects makes them good candidates for the study of emergent social structure and its relationship to technical activities.

The email discussions span a range of technical topics. In addition to discussing the source code (and specific entities such as functions and methods that occur in source code), the mailing list participants also discuss the build system, documentation, and high level architecture. But, even in OSS, few, if any developers work on the entire system; most specialize. Consequently, just as development teams are split up and modularized in a software company, we believe that within the entire cohort of developer mailing list participants, there are self-organizing teams that form as people organically tend to focus towards specific topics, subsystems, or tasks. We can use archives of the developer mailing lists to determine which individuals were in communication with each other to create a social network of the participants. But how can community structure be discovered in this social network?

In 2002, Newman and Girvan introduced a quantitative notion of the *community structure* of a network, as "the division of network nodes into groups within which the network connections are dense, but between which they are sparser" [24]. Fig 1 is an example of a network with strong community structure. Community structure has been investigated in many types of networks in recent years [2, 35, 51, 44, 60, 26, 3, 28, 56] due to advances in methods of identifying these structures [63, 64, 49, 48, 46, 13, 47]. We hypothesize that this kind of structure exists within the development communities of these large OSS projects. Mailing list participants spontaneously form subgroups, and communicate more intensively with people in subgroups than outside them. This leads us to our first research question:

Research Question 1 (RQ1) – *Is there evidence of latent community structure in the form of cohesive subgroups within the social networks of OSS projects?*

While we believe that groups of participants do organically form, this result is not interesting or of value unless the division into groups is related in some way to the purposes of the community. The communication and coordination activities on the development mailing lists serve a number of different purposes. While the most common purpose appears to be discussion of actual development activity (function interfaces, APIs, bug fixes, feature implementation, etc.), other topics include policy decisions, high level architectural changes, release plans, licensing issues, and votes on whether certain participants should be given write access to the repository. Broadly (and not entirely accurately) we call the former *product* topics, and the rest *process* topics. While the process topics are clearly vital to the success and continued life of the projects, they are less directly related to coding activities than product topics; as such, the division of knowledge issues that arise with large, intricate software system is not as critical in process topic discussions; everyone can know enough to gainfully participate. In fact, since everyone is affected by process issues, all *should* participate. Therefore, the social networks of participants in process discussions should not be fragmented or modularized. This leads to the conclusion that the social network fragmentation into subgroups will be more strongly evident in product-related discussions.

Research Question 2 (RQ2) – *Is the community structure* stronger *when considering only product-related discussions than when considering other discussions?*

We posit that if there is strong community structure within the projects, the groups should be related to the software engineering activities in a meaningful way. Specifically, we would expect that people within a group collaborate more closely than people in different groups. We can state this as a research question in two different ways. First, we can ask if people within the same group are more likely to collaborate directly, i.e, whether they work on the same files. Thus, the third research question that we address is:

Research Question 3 (RQ3) – *Are people within a subgroup more likely to collaborate directly than the people in different groups?*

Second, we can ask whether the groups are somehow allied with coherent tasks. For example, we can check whether the cumulative engineering effort of the people within a group tends to be a confined part of the system. Our final research question is thus:

Research Question 4 (RQ4) – *Are the software engineering activities of people within a subgroup organized around a cohesive task?*

We present our methods and processes for answering these questions, give the results of our analysis, and discuss these results in this paper. The rest of the paper is organized as follows. In section 3 we discuss other research that has examined the organization of OSS communities and include the differences and similarities to our work. Our datamining and analysis methods along with discussions of results are given in section 4. We examine the strengths and weaknesses of our approach in section 5. A synopsis and further areas of study appear in section 6.

Relevance to software engineers These questions above do relate to organizational science issues concerning the nature of open-source social structures. But they are also

[1]By developer, we refer to contributors that have write access to the source code repository

of deep concern to software engineers, for several reasons. First, all the projects we studied are both complex and highly successful. Strong evidence of sub-community formation in these projects is arguably prescriptive for any new and growing project; open source project leaders might do well to encourage subcommunity formation. Second, strong evidence of sub-communities forming around product-related activities (RQ2) suggests that newcomers aspiring to gain developer privileges [20, 8] might do well to find and connect with the right community; likewise, RQ2 might also suggest that the broadest possible participation is good for process-related issues. Finally, RQ3 and RQ4 might help inspire recommender systems [54] that find technically relevant people that one should connect with, in a large team. It should be noted that all these also have implications for the organizational and social structure of commercial software projects. Consequently, many software engineering researchers have studied related socio-technical issues [40, 31, 43, 62]. In fact, a recent invited talk at ICSE describes the importance of issues facing socio-technical coordination in a global environment (a key facet of OSS)[30].

3. RELATED WORK

Communication and Co-ordination behaviour in distributed teams has been studied. Ehrlich *et al* [22] studies how individuals in global teams acquire expertise. Layman *et al* [37] studies how a globally distributed team overcomes communication challenges to become agile. All of the above studies are concerned with commercial organizations. There is a significant body of research devoted to examining OSS communities, since the data is archived and more easily available.

Nicolas Ducheneaut [20], presented a case study of the Python community, examining the process by which an individual evolved from a casual mailing list peruser into a strong core developer. Ducheneanu's insights into community processes has been followed by other works [6, 7].

Social networks among developers have been studied from perspectives differing from ours. Xu *et al* [61] consider two developers socially related if they participate in the same project. We consider contributors related if there is evidence of email communication; this is arguably a more direct evidence of a social link. In addition, we examine each participant on the mailing list, not just the developers. Wagstrom, Herbsleb and Carley [58] gathered empirical social network data from several sources, including blogs, email lists and networking web sites, and built models of their social behavior; these models were then used to simulate how users joined and left projects. Our approach is empirical rather simulation-based. Crowston and Howison [16] use co-occurrence of developers on bug reports as indicators of a social link. They present evidence that the social networks of smaller projects are more central than those of larger projects. Presumably larger projects decentralize, to simplify communication and coordination activities. This observation is one of the key motivators behind this work as we hypothesize that subcommunities form naturally in larger, more complex and longer-lived projects.

Commit behavior in versioned repositories has been used as an indicator of social linkage. Lopez-Fernandez *et al* [38] consider two developers to be linked if they collaborate, *viz.,* they commit to the same module. The resulting social networks are similar in structure to ours. The work of De Souza *et al* [14] 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 question naturally arises, *How does commit behavior relate to knowledge exchange through direct social interaction?* The relationship between the two is a subject of our current research.

In previous work [6, 7] 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.

A number of researchers have used recently developed methods to find community structure in existing networks. Guimera *et al* [28] mined email logs from a company to create a social network and identified the community structure contained in it. They discussed the results as the *informal networks* behind the formal chart of an organization and it's benefit as a management tool.

González-Barahona, López and Robles have examined the community structure of modules within the Apache project [27]. In their analysis, they created a network with the vertices representing modules. Edges between vertices represented work on both modules by a common author and were weighted based on the number of commits the common authors had contributed. They examined the community structure of these networks over time and were able to see how the modules evolved with respect to each other.

Modularity, especially as it results from evolution and bestows benefits to an organism, is very important to the study of biological networks. In the areas of systems biology and bioinformatics authors have used Newman and Girvan's [33] as well as related [63] and other [64] algorithms to resolve modules in biological networks of different types. Using such algorithms, recent studies [33, 35] have shown that despite having evolved through random processes, biological networks exhibit design patterns, most notably high modularity. And although modular designs are not the most efficient when it comes to performing the day-to-day business in the cell [2], modularity apparently endows the networks, and hence the organisms, with systemic properties like robustness and evolvability [33, 35], which are essential for their long-term survival and fitness optimization [33].

An important issue is the relationship of social connections or communities and technical connections or communities. There is a great deal of current interest in *socio-technical congruence* (STC). The idea is that great alignment between communication patterns and task dependencies (either goal-precondition relationships of tasks, or data/control dependencies between artifacts) leads to better outcomes. Cataldo [12] studies the connection between task dependencies and co-ordination efforts by engineers. Valetto *et al* [57] suggest a formal, general, graph-based technique to measure congruence. We wish to study if subcommunities form within OSS projects, and whether STC appears to be a factor in their formation.

4. METHODS AND ANALYSIS

Our experiment involved several steps. We first identified

Name	Apache	Ant	Python	Perl	PostgreSQL
Begin Date	1995-02-27	2000-01-12	1999-04-21	1999-03-01	1998-01-03
End Date	2005-07-13	2006-08-31	2006-07-27	2007-06-20	2007-03-01
Messages	101250	73157	66541	112514	132698
List Participants	2017	1960	1329	3621	3607
Files	1092	7682	4290	13308	6083
Developers	57	40	92	25	29
Commits	28517	58254	48318	92502	111847

Table 1: Information on the data gathered for the projects studied.

the projects of interest and mined the developer mailing list archives and source code repositories of each of the projects. Next, we filtered the mailing list messages and created a social network of the participants over 3-month intervals. We then calculated the community structure of each social network. Following that, the relevance of the divisions of participants was evaluated quantitatively using mined source code development data and qualitatively by manual methods. The following subsections contain the pertinent details of each phase.

4.1 Project Selection

Our study includes the Apache webserver, Ant, Python, Perl, and PostgreSQL. These are all well known and stable projects. Each has undergone a number of major release cycles and is still under active development. Each has a developer mailing list with thousands of participants. All have large and complex codebases with several subsystems, making it difficult for any one person to be an expert on all parts of the system. This leads to a need for "division of labour", which we believe instigates the formation of subcommunities within these projects. In addition, email and source code revision archives, dating back several years, are publicly available. Table 1 shows the date ranges for the data gathered from each project as well as the numbers of messages sent, participants on the mailing list, files in the repository, developers with repository access, and source code repository commits.

We also have picked projects that vary in their *governance structure*. Some of these projects have been described by Berkus [5] as archetypes of very different governance styles. Both Apache projects (the webserver and Ant) are *foundations* with well-organized, hierarchical governance structure and formalized policies. PostgreSQL is a *community*, which is more informal and has a consensual group decision making process. Python and Perl are both *monarchist* with a project leader (Guido Van Rossum in the case of Python and Larry Wall for Perl) at the helm making informed important decisions. With this variety, we hope to ameliorate the threats to external validity.

4.2 Mining the Raw Data

The public email archives were downloaded and parsed into relational tables. For email, we extracted the date, the body, the name and email address of the sender, the *message-id* header, and the *in-reply-to* header. The last two are used to reconstruct threads of conversation. If the *message-id* of message A appears in the *in-reply-to* header of message B, then B was sent in response to A which indicates that the sender of B found message A "interesting". This "interest" may be a suggestion, rant, praise, disagreement, etc.; regardless, it is indicative of communication between the two parties. We thus create a link between the sender of A and the sender of B in the social network. Unfortunately, the accuracy of this network is compromised by the practice of *email aliasing*, whereby one mailing list participant uses several email addresses. To resolve the aliases, and identify and group email aliases, we use a range of techniques, including fuzzy string similarity, domain name matching, clustering, heuristics, and manual post-processing.

Once email aliasing is handled, for further processing, we analyze a time-series of the social networks, at 3 month intervals. For reasons that will become clear, we use an adjacency-matrix representation of the social network at each time interval.

In addition, we also extracted code information: the author, time of commit, the filename, and the contents of each file from the project source code repositories. The email addresses that corresponded to each repository author were also heuristically determined and hand verified in order to match the development activity and communication behavior of project developers. By using this commit information, we can see which developers were collaborating and on which files. Details of the email and repository mining processes can be found in our prior work [6].

4.3 Finding Community Structure

To find and quantify the latent community structure that exists in the OSS networks, we have created a variant of the Newman algorithm[2] [47].

The goal is to partition the network into groups of nodes, so the connections within groups are dense and the connections between the groups are sparse. Newman and Girvan defined a *modularity measure*, which quantifies community structure strength, using the denseness and sparsity of the groups' intra and interconnections [49]. Consider 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

$$Q = \sum_i (e_{ii} - a_i^2) \quad (1)$$

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 (that is, the same division on a random network with the same degree distribution). Values for Q range from 0 (with networks of essentially random structure) to 1. In naturally occurring networks that are known to be

[2]We gratefully acknowledge Mark Newman's help in giving us a source code implementation of his algorithm as a starting point.

strongly modular, modularity measure values ranging from 0.3 to 0.7 have been observed [49]. The algorithm also has been shown to find modules known *a priori*. In our case, partitioning the social networks, we want to find the partition that yields the highest modularity for the network. Finding the partition that maximizes the modularity for a given network is an NP-complete problem [18, 23]. Newman & Girvan's method is approximate, but empirically effective. See [29] and [10] for examples.

Girvan and Newman's original algorithm works well for binary networks, but doesn't handle networks with weighted edges. Our social networks contain weighted edges, representing the number of emails exchanged between two participants in each time period. A high number of messages between a pair of participants should increase their likelihood of being in the same group. Following a method for adapting binary network algorithms to work on weighted networks [45], we modified our social networks by introducing one edge between each pair of nodes per email sent between them (i.e. creating a multi-edge network) and modified Newman's algorithm above to handle multi-edge networks.

4.4 Filtering Messages

As we applied the community structure identification techniques described above to the OSS projects, we examined the activities of the members in the various subcommunities by hand to see if we were identifying meaningful partitions of the project community. Two key observations arose out of these manual inspections which led us to refine our process.

First, although it is very hard to have an intimate knowledge of all parts of a complex system, there do appear to be a very small number of people in each project that actually do have at least a working knowledge of nearly all of its parts. These people are usually the project leader, the founding member, or very early project members and can be identified by the quantity, quality, and broad spectrum of commits to the repositories and their discussion of all aspects of the software on the mailing lists. Examples of these people include Tim Peters and Guido Van Rossum in Python, Bruce Momjian and Tom Lane in Postgres, and William Rowe and Jeff Trawick in Apache, Most of these contributors have been members of the project for a very long time, have high social status within the project, are within the "inner circle" of elite developers, and often comment in nearly every mailing list thread. These individuals tend to play leadership roles in the project, and are concerned with a broad set of technical issues. Given their broad interests, we would expect that these leaders would work with different specialized groups, and thus not contribute to modularity. In some sense, these folks play the role of managers in traditional hierarchically decomposed, deliberate organizational structures. Just as managers tend to serve as a focal point for *inter*-group co-ordination and communication, these OSS leaders co-ordinate the activities of the OSS developers. Since these people tend to connect to many others in the community, they will prevent any latent sub-communities that may exist from being detected by automated algorithms, even modern, powerful ones like Newman's. We therefore remove them consideration while seeking sub-communities.

In previous work, ourselves and others have found that *betweenness centrality* [59] is indicative of high levels of social status, power, and managerial roles in both open source [6] and commercial [1, 34] contexts.

We use this form of link analysis to determining these people with high social status and source code contribution and manually examining their activities. So as not to bias our results too much, we never remove more than three of these participants.

Second, we found that the interaction patterns of people were different when discussing *product* versus *process* related topics. In order to identify *product* topic discussions in the modularity analysis, we use source code analysis to identify messages which reference the source code. Key terms were extracted from all version of the source code in the repository using the the static analysis tool, Understand, from Scitools. These key terms include variable names, functions, classes, and filenames.

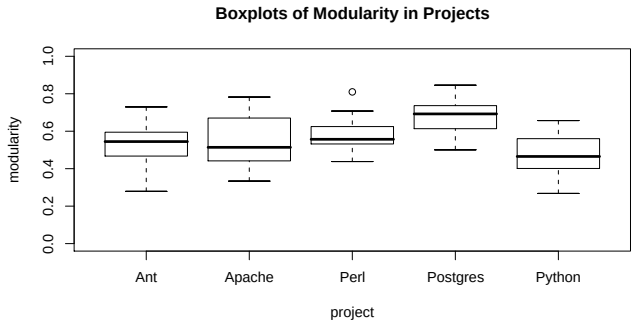

Figure 2: Boxplots of the strength of community structure for the various projects studied.

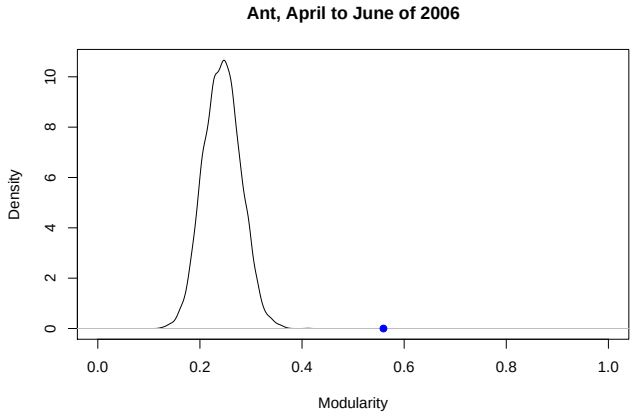

Figure 3: The distribution of modularity values for random graphs with the same degree distribution as the observed network. The point represents the actual observed value.

We found strong levels of community structure in all of the projects studied. The value of the Q measure, as defined in (1), ranges from 0.4 to 0.8. The range of values for different projects, over the studied period, is shown in Figure 2. To concretize this scalar value, we show in Figure 4

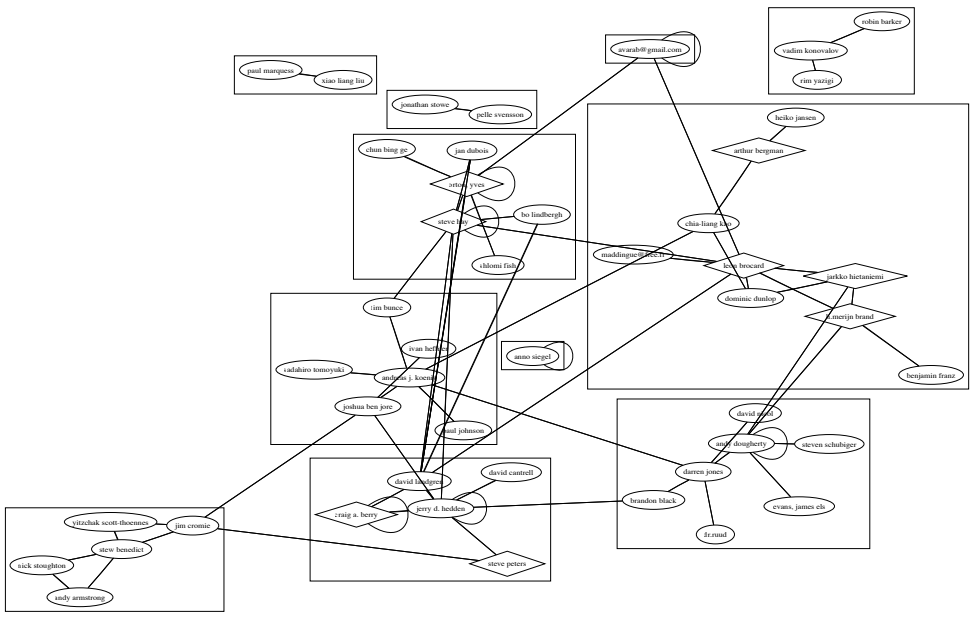

Figure 4: The community structure of Perl from April to June 2007. Diamonds indicate actual developers. Edge weights are not depicted.

an example of a network with a community structure value of 0.76 that is taken from the Perl project for the months of April to June of 2007. This example was chosen because of its relatively small size in relation to the other time periods and projects studied[3]. In Figure 4, an edge represents one or more messages between participants; edge weights, albeit used by the algorithm, are not depicted graphically. Several distinct subgroups can be seen; typically the edges within subgroups represent frequent communications. Newman has found that in naturally occurring networks, modularity values of 0.3 and above indicate strong community structure [49]. As can be seen in Figure 5 we we found values in this range both before and after filtering messages.

Significance of Observed Modularity: The question, arises, are these values of modularity statistically significant? To paraphrase the question, do the empirically observed modularity values reflect something special and real about how people associate and communicate on the observed email social networks, or are they just values that would arise in any random network where the same people were equally active, but had different associations? If the latter is true, that would suggest that *who* people talk to doesn't matter, only *how much* they tak. Our claim, however, is that subcommunities form because people deliberately choose who they communicate with.

We assess this question by comparing the observed values with those obtained by analyzing random graphs sharing similar properties [42, 50]. Here we want to see if people associate into subgroups in a statistically significant way. Therefore we randomize networks by assuming that people on the network remain equally active, *i.e.,* send just as many messages, but send them to a randomly chosen group of people, rather than selectively associated with some. We generated a large number[4] of random graphs with the *same degree distribution* as the observed networks using a rewiring approach [19, 39, 25]. This technique begins with the empirical network and randomly chooses pairs of edges whose endpoints are flipped. The edges are "rewired" in this manner until a sufficiently random graph is generated. As with the observed networks, we removed the three highest betweenness nodes to make the comparisons fair.

A comparison of modularity values from the random networks with those from the actual networks can reject the null hypothesis at far below the .001 level. An example of a modularity distribution for Ant from April to June of 2006 is shown in Figure 3. The point on the right indicates the observed network and the curve shown is the distribution of modularity values obtained from random networks with the same degree distribution. Therefore we reject the null hypothesis that the observed modularity values would occur in a random network where individuals were equally socially active. Therefore we conclude that **RQ1** *is confirmed.*

4.5 Effect of Product and Process Topics

While we did identify strong community structure in the social networks prior to this filtering step, more clearly delineated subcommunities emerge when constraining the communication that we use in our analysis to messages directly mentioning product topics, *viz.,* emails that specifically name actual code artifacts.

As an example, figure 5 shows the modularity found in the PostgreSQL project over time when using all messages on the developer mailing list and when using only messages that directly mention functions or file names.

Table 2 shows the average increase in modularity that we obtain when we include only the *product* topic emails. We examined the differences between the filtered and unfiltered values using paired Wilcoxon tests.

To asses the statistical significance of the results, since we are testing multiple hypotheses (5 in this case), the individ-

[3]Graphs of the networks for each time period of each project can be viewed at `http://janus.cs.ucdavis.edu/~cabird/cs-graphs`.

[4]roughly 30,000 per observed network

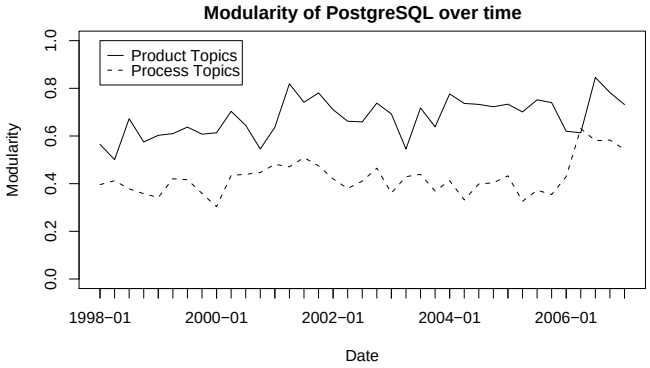

Figure 5: The difference in strength of community structure in the PostgreSQL project over time when filtering on messages that include product-related terms.

Name	Apache	Ant	Python	Perl	Postgres
Product	0.548	0.534	0.473	0.567	0.679
Process	0.337	0.485	0.312	0.423	0.425
All	0.325	0.459	0.293	0.400	0.420
P-val	0.001	0.033	0.001	0.001	0.001
%*Product*	27.6	64.3	50.0	29.9	26.4

Table 2: Means of the modularity when examining only *hard* emails, only *soft* emails, or all emails. P-val represents the statistical significance of a paired Wilcoxon test of *product* and *process* populations per project. The bottom row is the proportion of messages (as a %) that relate to product topics

ual p-values during testing were adjusted using Benjamini-Hochberg adjustment for multiple hypotheses [4]. This procedure maintains an overall false positive rate of below 0.05 (this is known as the False Discovery Rate). The results were statistically significant with p-values below .05 in all cases.

Note that an increase in modularity when filtering the edges in a network is not a foregone conclusion. Rather, we did not see an increase in modularity when examining only the *process* emails relative to all emails. A comparison of the modularity based on *process* and *product* topic emails in addition to the entire network (labelled "All") is shown in Table 2. This indicates that the groupings into subcommunities is much stronger when discussions directly related to the source code arise. Thus **RQ2** *is confirmed*. This affirmative answer to **RQ2** suggests that successful projects tend to focus into subgroups for product-related work, but discuss process-related issues more broadly.

4.6 Community Structure and Collaboration

We now present the first of two experiments conducted to validate whether the subgroups discovered by the algorithm reflect groups that are technically cohesive in some way (specifically, Research Question 3) First, in OSS projects, communication activities are related largely to developer effort. If two individuals are directly collaborating on the same artifact, we would expect them to coordinate their activities, and thus be strongly connected. Therefore they should organize themselves into the same subcommunity. We concretize this concept by specifically testing whether developers within a subcommunity will be more likely to work on the same files than developers in different subcommunities. This can be tested quantitatively.

Let D represent the set of developers that are active in a given time period for a project. Let $g(x)$ represent the group of developer x and let $f(x)$ represent the set of files modified by developer x for the same time period. Now define two populations P_{same} and P_{diff} in the following way. For every pair of developers, $x, y \in D$ if $g(x) = g(y)$, add $|f(x) \cap f(y)|$ to P_{same}, and if $g(x) \neq g(y)$ add $|f(x) \cap f(y)|$ to P_{diff}. P_{same} represents collaboration between developers in the same group and P_{diff} represents collaboration between different groups. Since the majority of pairs of developers don't work on any files together, neither of these populations are normally distributed, making a t-test inappropriate [9, 17]. We therefore use a two sample Mann-Whitney test (a nonparametric test also known as a Wilcoxon test) to test the difference in means between the populations. If developers in the same group are more likely to collaborate on files together, the mean of P_{same} will be higher than P_{diff} to a statistically significant degree. We therefore perform this test on all of the projects studied.

Results: We found that in four of the five projects, (See Table 3) developers worked together on the same file with people in their own group much more often than people in others on average. We show the p-values for both the Wilcoxon Rank Sum test (also known as a Mann-Whitney test) in the first column, and the Kolmogorov-Smirnov test [55], which examines the two distributions for each project to determine if the same group distribution was significantly higher that the different group distribution, in the second column. These p-values were also adjusted for multiple hypotheses testing. The results are generally statistically quite significant (same group distribution was significantly higher that the different group distribution) in the second column.

Unfortunately, in the case of Perl, while we were able to access repository logs, we were unable to obtain the actual repository files and therefore could not run our static analysis tools on them to get function names, The key terms for Perl were limited only to the filenames in the repository. Therefore our experiment on Perl was incomplete, and our results are inconclusive; however, since we did try it, we left the data in table.

Project	Wilcoxon P-val	Kolmogorov-Smirnov P-val
Ant	0.000	0.000
Apache	0.052	0.001
Perl	0.503	0.842
Postgres	0.000	0.000
Python	0.000	0.000

Table 3: Probability values for non-parametric tests of difference in means and difference in distributions of co-commits of developers between groups and within groups corrected for multiple hypothesis testing. The data on Perl was incomplete, so the resulting p-value is inconclusive.

We therefore conclude that for the Ant, Apache, Postgres and Python projects, since developers have higher collaboration levels with other developers in their own group than with developers outside of their group, the community structure of the social networks does hold relevance to the actual development effort. *Thus* **RQ3** *is confirmed*. This suggests

that in successful projects, co-commit behaviour is strongly linked with social interaction.

4.7 Community Structure and Activity Focus

In addition to increases in collaboration, we also hypothesize that developers within the same group will be more likely to work within specific areas or subsystems within the code base. In order to quantify this "scope of activity", we examine the average directory tree distance between all pairs of files that are committed to by developers within each group (weighted by the number of commits). Smaller distances between pairs of files for a given group of developers indicates smaller scope and more focus. This methodology is based on the common (albeit not universal) practice of basing the directory structure on the architecture of the system. The null hypothesis is that the weighted average distance between all pairs of files committed to by developers in the same group will be no different than for randomly drawn sets of developers that are the same size as the group. We expect that the directory tree distance between committed files will be smaller for groups of developers from the same subcommunity. We therefore compare the observed average distances between all pairs of files committed to by developers in one group to a large number of runs where the set of developers was chosen randomly.

After performing this analysis, we were unable to reject the null hypothesis (no difference in directory distance) for any of the projects. **RQ4** *is therefore not quantitatively confirmed* using the above, directory-tree distance method.

Although there were cases where the average distance for files from a group of developers was far smaller than the average for all tests of random sets of developers, the trend was not consistent throughout. There are two possible reasons for this inconclusive result; either the hypothesis is incorrect and the groups did not have a specific task focus, or the precise definition and nature of "task focus" that we used was lacking. In order to shed light on the matter, we mounted a case study to try understand the topics of discussion, and the commit behaviour, of developers in sub-communities.

Case Studies We carefully studied the emails on developer lists and commits to files in source code repositories. This information represents the actual work that goes on in the projects on a daily basis. We therefore examine this data for the groups of participants identified by the community structure algorithms. Our goal was to find if there were common tasks, topics, or particular subsystems in the activities of participants in subcommunities. We have identified time periods and subcommunities where these indicators have emerged and discuss examples of these here. We found that the subcommunities can be categorized into three types.

We also examined the development and communication activities of people in the groups identified to see if they were in fact working together on common tasks. Due to the sheer number of groups identified over the life of all five projects, a comprehensive manual inspection was not possible. We therefore studied a few cases where the work of groups seemed strongly focused on one part of the directory structure, and cases where it seemed strongly *un*focused. These cases were quite instructive.

A sub-community in Apache The first category of community is that in which the discussion and development is focused on one are of the codebase. In the Apache webserver project, from May to July of 2003, one subcommunity consisted of Rowe and Thorpe (developers) as well as Deaves, Adkins, and Chandran. They discussed some bug fixes to `mod_ssl`, the apache interface to the Secure Sockets Layer (SSL). Topics included issues with incorrect input/output code, module loading, unloading and initialization, and integration of `mod_ssl` with the server. Nearly all the discussion is about the SSL code, and virtually all of the files modified by people in this group during this time period are in the `modules/ssl` directory. Clearly, this is a subcommunity within the Apache project that is focused on a particular task. In other cases, groups of participants were not focused on one single topic or task. Often this would occur when one or two developers worked on two or more disparate areas of the code base, thus drawing two communities together. There were also a smaller number of cases where no clear topics were distinguishable from the changes to files or the email messages. Many of these occurred relatively close to release dates.

A sub-community in Python The second type of subcommunity had a clear focus in both discussion and content of development behaviour, but the locations of the modified files cross-cut across the directory structure. From April to June of 2003, the Python developer mailing lists provides an instructive illustration of this phenomenon. The key participants in one identified group of the python community are Hylton, Cannon, and Fulton. During this time Hylton was diagnosing memory leaks in Zope, an object oriented web application server written in python[5]. Using unit tests, Hylton tracked down problems in the garbage collection code. There are several related messages on the mailing list. Fulton, Paul Prescod, and Hylton discuss the use, semantics, and expected behavior of the garbage collection API's on both the C and Python parts of the code base, with example code of their use posted. The discussion results in several changes in the files relating to garbage collection. Extensive changes are made to `/Modules/gcmodule.c` and in other areas of the python interpreter such as function objects in `/Object/funcobject` and handling of pickled objects in `/Modules/cPickle.c`. The tracing and inspection modules of the python interpreter are also modified to enhance future debugging of the GC code. In addition, while testing this code, a few of the unit tests fail and discussions ensue between Hylton and Cannon, resulting in diagnosis and remediation of bugs in the unit test code *per se*. Changes were made to `urllib2.py, httplib.py` and `strptime.py`, *inter alia*. Figure 6 is a snapshot of this group of contributors along with the directories that they committed to. Diamonds indicate developers, ovals are participants, and rectangles are directories. Clearly, this group is operating as a team to accomplish a common goal: improving the quality of the garbage collector. However, the issue dealt with has parts scattered across the code base. Clearly, the the garbage collection code is a concern that cuts across the module and directory structure, *i.e.* it is an aspect. This leads to the interesting observation that even if a feature is cross-cutting, affects a broad swath of files, the discussion surrounding it may be cohesive, and involves a well-defied sub-community of developers. This suggests in fact, an alternative approach to aspect-mining, based on finding apparently unrelated files that consistently worked

[5]For details, see `http://www.python.org/~jeremy/weblog/0304.html`

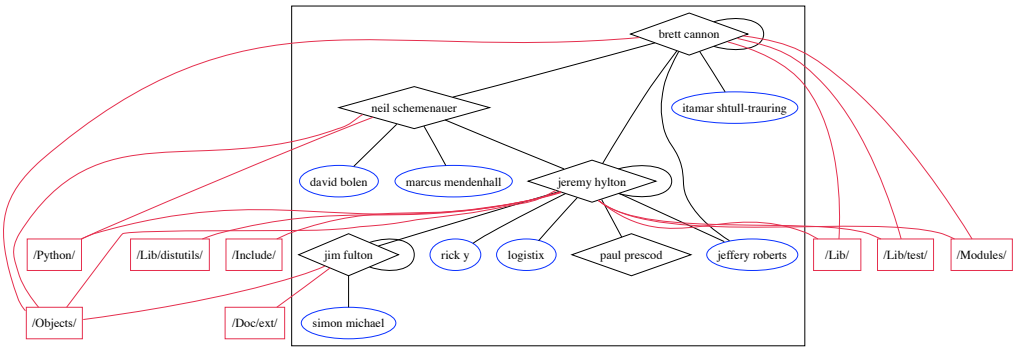

Figure 6: One group of participants in the python community from April to June 2003. Diamonds represent developers, ovals are participants, and rectangles are directories committed to (in lieu of the large number of files committed to).

on by groups of developers with strong social ties.

Sub-communities in other projects We studied several sub-communities in each of the projects studied, and generally found good evidence for task focus.

For example, 11/2002 to 12/2002 in postgres, one subcommunity works solely on embedded SQL in C, and another focuses on updating the SGML documentation source. In the following time period, a group emerges whose activity and discussion concerns the development and testing of the postrges JDBC driver (with source code and test code spanning the code base within the JDBC subtree) and another much smaller group works on Unicode support.

There are other sub-communities whose focus is not as localized within the system. During 10/2001 to 12/2001, we find at two sub-communities whose tasks broadly span the Ant codebase. One large group, with 29 participants (including 5 developers) focuses on tracing and debugging. Although their code modifies files in many different places, their changes broadly add logging calls, debugging output, and assertions. Another group of 9 participants and 3 developers is working on ant build tasks, many specific to other (non-ANT) and commercial software, including EJB, VisualAge, Perforce, JUnit, and Sitraka products. The affected files are scattered in different packages. In addition, many test cases, in an entirely different part of the system's directory structure are also updated.

In the third type of subcommunity there was more than one particular topic or task under discussion or development. Often this would occur when one or two developers worked on two or more disparate areas of the code base, thus drawing two communities together. We also noted a few cases where no clear topics were distinguishable from the changes to files or the email messages (a number of these cases occurred relatively close to release dates).

In conclusion, **RQ4**, the question of whether sub-communities focus around cohesive tasks is clearly a complex one. Sub-communities sometimes relate to closely related files in a single module, and sometimes they don't. Our case study indicates one possible explanation—perhaps, in some cases, the focal task relates to cross-cutting concerns. We are currently exploring this issue, specially as it relates to the possibility of automatically mining socially/conceptually coherent, cross-cutting concerns; this might suggest either refactoring, or introducing the use of Aspect-oriented programming.

5. THREATS TO VALIDITY

For our analysis, we generate social links between developers only from the developer mailing list. While the mailing list is the proscribed location for discussion (due to it's broadcast nature) [6, 20, 32, 41], we miss communication that may occur directly between developers such as in private emails or on irc channels, or other communication channels such as comments in bug reports.

While there appears to be a relationship between development activity and community structure, it is important to note that no causal link has been established. Further work is required to determine if the social links drive collaboration or vice versa, (or if they are both results of an unobserved phenomenon).

The biggest threat is to external validity. As with most studies of open source software, the projects for study were chosen based on certain criteria as mentioned in section 4.1. This necessary bias in selection means that we are not randomly sampling from the population of OSS projects. Therefore, while these results may be similar to what occurs in projects that don't fit these criteria, we have no evidence to support that assertion. In addition, as an examination of just five projects, these results may not generalize even to other projects that fit the same criteria. We believe that they would, but some have argued otherwise [5].

6. CONCLUSION

We have mined the communication and development data for five large open source projects and tailored an algorithm to examined their email social networks for evidence of teams whose activities are directly related to the software artifact. We found that in all cases, evidence of strong community structure existed within the communication patterns of the participant. In addition, in all cases where our data was complete, the division of the project into subcommunities was also representative of the collaboration behavior of the developers. A quantitative analysis of the task focus for the various teams was inconclusive, but some case studies indicated that the task focus of subgroups does exist in many cases, but can be subtle and varied in nature. The case study suggests some directions for future work, both in socio-technical congruence, and in aspect-mining.

7. ACKNOWLEDGEMENTS

We would like to thank SciTools[6] for graciously allowing us the use of their excellent static analysis tools for Java, C, and C++. We also gratefully acknowledge support from the National Science Foundation Science of Design program, *NSF-SoD-0613949*

8. REFERENCES

[1] Ahuja, Manju K., Galletta, Dennis F., and Carley, Kathleen M. Individual centrality and performance in virtual r&d groups: An empirical study. *Management Science*, 49(1):21–38, jan 2003.

[2] U. Alon. Biological Networks: The Tinkerer as an Engineer. *Science*, 301(5641):1866–1867, 2003.

[3] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54, 2006.

[4] Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.

[5] J. Berkus. The 5 types of open source projects. March 20, 2007 `http://www.powerpostgresql.com/5_types`.

[6] 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.

[7] 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.

[8] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? immigration in open source projects. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 6, Washington, DC, USA, 2007. IEEE Computer Society.

[9] G. Box, W. Hunter, and J. Hunter. Statistics for experimenters: an introductory to design data analysis and model building. *Wiley Series in Probability and Mathematical Statistics).*, 1978.

[10] P. Boykin and V. Roychowdhury. Personal Email Networks: An Effective Anti-Spam Tool. *Arxiv preprint cond-mat/0402143*, 2004.

[11] F. Brooks. *The mythical man-month.* Addison-Wesley, 1995.

[12] M. Cataldo, P. Wagstrom, J. Herbsleb, and K. Carley. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362, 2006.

[13] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):66111, 2004.

[14] 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`.

[15] M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[16] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[17] P. Dalgaard. *Introductory Statistics With R.* Springer, 2002.

[18] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 9:P09008, 2005.

[19] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW.* Oxford University Press, 2003.

[20] N. Ducheneaut. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, 2005.

[21] N. Ducheneaut and L. Watts. In search of coherence: a review of e-mail research. *Human-Computer Interaction*, 20(1-2):11–48, 2005.

[22] K. Ehrlich, K. Chang, I. Res, and M. Cambridge. Leveraging expertise in global software teams: Going outside boundaries. *Global Software Engineering, 2006. ICGSE'06. International Conference on*, pages 149–158, 2006.

[23] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman & Co. New York, NY, USA, 1979.

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

[25] C. Gkantsidis, M. Mihail, and E. Zegura. The markov chain simulation method for generating connected power law random graphs. In *Proceedings of ALENEX '03*, pages 16–25, 2003.

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

[27] J. González-Barahona, L. López, and G. Robles. Community structure of modules in the apache project. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, 2005.

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

[29] R. Guimerà, S. Mossa, A. Turtschi, and L. Amaral. From the Cover: The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proc Natl Acad Sci US A*, 102(22):7794–7799, 2005.

[30] J. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. *International Conference on Software Engineering*, pages 188–198, 2007.

[31] J. D. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *ESEC / SIGSOFT FSE*, pages 138–137, 2003.

[6] `http://www.scitools.com`

[32] G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7):1159–1177, 2003.

[33] A. Hintze and C. Adami. Evolution of complex modular biological networks. *PloS Computational Biology, e23.eor*, 2008.

[34] H. Ibarra. Network centrality, power, and innovation involvement: Determinants of technical and administrative roles. *The Academy of Management Journal*, 36(3):471–501, jun 1993.

[35] N. Kashtan and U. Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102(39):13773–13778, 2005.

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

[37] L. Layman, L. Williams, D. Damian, and H. Bures. Essential communication practices for Extreme Programming in a global software development team. *Information and Software Technology*, 48(9):781–794, 2006.

[38] 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.

[39] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *Arxiv preprint cond-mat/0312028*, 2003.

[40] A. Mockus, R. Fielding, and J. Herbsleb. A case study of open source software development: The Apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 263–272, Limerick, Ireland, 2000.

[41] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[42] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6(2-3):161–179, 1995.

[43] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. *Proceedings of the International Workshop on Principles of Software Evolution*, pages 76–85, 2002.

[44] M. Newman. Coauthorship networks and patterns of scientific collaboration. *Proceedings of the National Academy of Sciences*, 101(suppl_ 1):5200–5205, 2004.

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

[46] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter*, 38(2):321–330, 2004.

[47] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):36104, 2006.

[48] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[49] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

[50] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64(2):026118, Jul 2001.

[51] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Arxiv preprint physics/0506133*, 2005.

[52] D. Parnas. The criteria to be used in decomposing systems into modules. *Communications of the ACM*, 14(1):221–227, 1972.

[53] 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.

[54] M. P. Robillard. Bellairs workshop on recommender systems, 3 2008.

[55] H. Scheffe. Statistical inference in the non-parametric case. *The Annals of Mathematical Statistics*, 14(4):305–332, dec 1943.

[56] J. Tyler, D. Wilkinson, and B. Huberman. E-Mail as Spectroscopy: Automated Discovery of Community Structure within Organizations. *The Information Society*, 21(2):143–153, 2005.

[57] G. Valetto, M. Helander, K. Ehrlich, S. Chulani, M. Wegman, and C. Williams. Using Software Repositories to Investigate Socio-technical Congruence in Development Projects. *Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007.

[58] P. Wagstrom, J. Herbsleb, and K. Carley. A Social Network Approach To Free/Open Source Software Simulation. *Proceedings of the 1st International Conference on Open Source Systems, Genova, 11th-15th July*, 2005.

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

[60] D. Wilkinson et al. A method for finding communities of related genes. *Proceedings of the National Academy of Sciences*, 101(suppl_ 1):5241–5248, 2004.

[61] 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.

[62] Y. Ye, Y. Yamamoto, and K. Nakakoji. A socio-technical framework for supporting programmers. *Proceedings of the 6th joint meeting of the european software engineering conference and the 14th ACM SIGSOFT symposium on Foundations of software engineering*, pages 351–360, 2007.

[63] J. Yoon, A. Blumer, and K. Lee. An algorithm for modularity analysis of directed and weighted biological networks based on edge-betweenness centrality. *Bioinformatics*, 22(24):3106, 2006.

[64] E. Ziv, M. Middendorf, and C. Wiggins. Information-theoretic approach to network modularity. *Physical Review E*, 71(4):46117, 2005.