Socio-Technical Developer Networks: Should We Trust Our Measurements?

Andrew Meneely, Laurie Williams
North Carolina State University
890 Oval Drive, Engineering Building 2, Campus Box 8206
Raleigh, North Carolina, USA 27695
{apmeneel,lawilli3}@ncsu.edu

ABSTRACT

Software development teams must be properly structured to provide effective collaboration to produce quality software. Over the last several years, social network analysis (SNA) has emerged as a popular method for studying the collaboration and organization of people working in large software development teams. Researchers have been modeling networks of developers based on socio-technical connections found in software development artifacts. Using these developer networks, researchers have proposed several SNA metrics that can predict software quality factors and describe the team structure. But do SNA metrics measure what they purport to measure? The objective of this research is to investigate if SNA metrics represent socio-technical relationships by examining if developer networks can be corroborated with developer perceptions. To measure developer perceptions, we developed an online survey that is personalized to each developer of a development team based on that developer's SNA metrics. Developers answered questions about other members of the team, such as identifying their collaborators and the project experts. A total of 124 developers responded to our survey from three popular open source projects: the Linux kernel, the PHP programming language, and the Wireshark network protocol analyzer. Our results indicate that connections in the developer network are statistically associated with the collaborators whom the developers named. Our results substantiate that SNA metrics represent socio-technical relationships in open source development projects, while also clarifying how the developer network can be interpreted by researchers and practitioners.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – process metrics, product metrics

General Terms

Measurement, Human Factors

Keywords

Developers, social network analysis, developer network

ICSE '11, May 21–28, 2011, Honolulu, Hawaii, USA.

1. INTRODUCTION

Behind most software products are teams of people collaborating with each other. Misguided effort can result in poor software quality, so large software development teams must often organize into a complex ecosystem of communication and coordination. An understanding of developer collaboration from the perspective of the entire team could help structure development efforts.

Over the last several years, social network analysis (SNA) has emerged as a popular method for studying the collaboration and organization of large software development teams. Researchers have been modeling networks of developers based on socio-technical[1] connections found in software development artifacts. Most SNA studies apply graph theory to socio-technical networks of developers (often called *developer networks*) to generate SNA metrics. Connections between developers in socio-technical networks often originate from software development artifacts, such as version control change logs. Recent empirical case studies of several well-known software products have shown that SNA metrics are predictive of faults [11, 13], failures [10, 18], and vulnerabilities [8, 9, 16]. Furthermore, researchers are developing visual tools [1, 15] and techniques [2-4, 7] to aid practitioners in viewing, analyzing, and organizing software development teams via developer networks.

While prediction studies and tool studies showcase the utility of SNA metrics, we must ask the question: do SNA metrics measure what they purport to measure? Both researchers and practitioners alike need to know the extent to which the developer network and SNA metrics represent the reality of a software development project. For example, if the version control logs show that two developers are working on the same code in the same month, are they collaborating? Are well-connected developers (i.e. *central* according to SNA metrics) viewed as experts in the project by teammates? The answers to these questions underlie many of the assumptions of SNA research and its recommendations.

While development artifacts can provide a historical view of the development project, the developers themselves can provide another valuable perspective. Developer perceptions could support that the connections observed in the development artifacts represent actual socio-technical relationships. A study of the comparison between developer perceptions and the developer network would help researchers and practitioners draw sound conclusions when analyzing a complex ecosystem of developers.

[1] We use "socio-technical" to describe the connection between two people in the context of work-related collaboration [6, 17]

Therefore, *the objective of this research is to investigate if social network analysis metrics represent socio-technical relationships by examining if developer networks can be corroborated with developer perceptions*. To measure developer perceptions, we developed an online survey that is personalized to each developer of a development team. The personalization is based on the SNA metrics taken from version control change logs of the developer's project. Developers answered questions about other members of the team, such as identifying collaborators and the project's experts.

A total of 124 responded from three open source projects: the Linux kernel, the PHP programming language, and the Wireshark network protocol analyzer. In this paper, we provide an empirical analysis of those responses and their relation to the developer network and its derived SNA metrics.

The main contribution of this paper is the empirical support that developer networks represent the real-world socio-technical concepts of collaboration, distance, and expertise. Specifically, we have found empirical support for all three of the following research questions:

- Q1: **Developer network edges**. If two developers work on the same source code file in the same month, then do they perceive they are collaborating?
- Q2: **Developer network distance**. Does the distance between two developers in the developer network represent the perceived degree of separation between those two developers?
- Q3: **Developer network centrality**. Does having a high centrality in the developer network indicate being reputed as a project expert by other developers?

The rest of this paper is organized as follows. Section 2 provides background on SNA. Section 3 provides related work. Section 4 defines our developer network for this study. Sections 5 and 6 describe our online survey and the three case studies, respectively. Section 7 describes the results of our analysis. Sections 7 and 8 discuss limitations and conclusions, respectively.

2. BACKGROUND

In this paper, we use several terms from network analysis. Network analysis is the study of characterizing and quantifying network structures, represented by graphs [5]. In network analysis, vertices of a graph are called **nodes**, and a connection between two nodes is called an **edge**. The **degree** of a node is defined as the number of adjacent nodes directly connected to the node.

A sequence of non-repeating, adjacent nodes is a **path**, and a shortest path between two nodes is called a **geodesic path** (note that geodesic paths are not necessarily unique). In the case of weighted edges, the geodesic path is the path of minimum weight. The **diameter** of a network is equal to the length of the longest unweighted geodesic path in the network. Informally, the unweighted geodesic path represents the "degree of separation" between two nodes.

Centrality metrics are used to quantify the location of a node or edge relative to the rest of the network. In this study, we use the **betweenness** metric to quantify the centrality of a node in a network. The betweenness [5] of node *n* is defined as the number of geodesic paths that include *n*. In this paper, we also normalize our betweenness by the total number of possible paths in the network. A high betweenness means a high centrality. Informally, if a node that lies "between" many other nodes is highly central.

3. RELATED WORK

Applying SNA to software development teams has been a heavily researched topic over the last several years. The work can be summarized in three different categories: prediction studies, internal validity studies, and tools or techniques for the purpose of studying developer communities.

The most prevalent of these three types of studies are the prediction studies [8-11, 13, 16, 18]. For example, in our previous work [8], we formed a developer network of the Linux kernel using the version control change logs. Other researchers, such as Pinzger [13], formed a variation of the developer network called the "contribution network". A contribution network is a graph with two types of nodes: developers and files. Edges exist between developers and files if a developer made a commit to a file. In contribution networks, no edges exist between developers or between files. The contribution network can be used to compute whether a file has been changed by developers who are changing many other files at the time (called an "unfocused contribution" [8, 9, 13, 16]) Studies in prediction have been performed on open- and closed-source products. Other case studies include the Jazz platform [18], Microsoft Windows [11, 13], a Nortel product [10], Mozilla Firefox [16], and the Linux kernel [8, 9, 16]. Together, the prediction studies have shown that SNA metrics are statistically correlated with external quality measures. The existence of this empirical evidence compels us to study SNA metrics directly to gain better insight into their validity.

We are not the first to conduct a validity study on SNA of development teams [7, 12]. Nia, et al. [12] discuss the validity of information flow in situations where not every edge of the developer network may be known. Their findings indicate that some centrality measures are sensitive to such a missing edge, and provide guidance on which measures ought to be used with developer networks. Meneely et al. [7] examined the connection between developer network centrality and the notion of "solution approver". Studying the OpenMRS healthcare project, the authors examined 602 issue reports and manually recorded the people who approved the final to solution to the issue. The authors formed a developer network from the version control change logs and found that central developers also happen to be solution approvers.

In addition to the prediction and validity studies, other studies have developed better insight into development teams using SNA in general. Bird et al. in studied the bazaar-like structure of open source projects [2, 4]. Specifically, the authors found that open source software teams self-organize into community structure, as evidenced by mailing list archives and version control change logs. Additionally, the authors find that developer networks found in open source projects reflect similar collaboration networks found in other disciplines. Also, Sarma et al. have developed a tool that visualizes many different aspects of development artifacts, including the developer network [15]. Lastly, Begel et al. have developed a tool at Microsoft that utilizes development artifacts to aid in finding people with specific expertise in a project [1].

4. DEVELOPER NETWORKS

The purpose of a developer network is to represent the complex system of socio-technical relationships between developers in a software development project. We borrow the term "socio-technical" from studies in psychology and sociology [6, 17] that discuss how teams of people can be arranged to work on a set of tasks. The term "socio-technical" refers to a labor-related connection among people that could be influenced by social or technical factors. The term "technical" is not referring to technology-related activities, but to the more general idea of technicality, skill, and labor. The idea of a developer network is to model socio-technical relationships explicitly with edges between developers in a network.

Software engineering researchers use a variety of approaches in creating their developer networks. In all cases, researchers are looking for records of social or technical connections in the context of the project. Most researchers [4, 7-12, 16, 18] use logs from the version control system (e.g. Subversion, Git) as a record of what code developers were working on. The motivation is that if a developer knows enough about the code to enact a change, then two developers changing the same code means a socio-technical relationship likely exists.

Some researchers [7, 18] apply communications logs (e.g. issue tracking comments, chat logs, or mailing list archives) to augment their developer network. We did not examine those artifacts for two reasons. First, not every case study we examined had communication archives, making a comparison across projects difficult. Second, communication archives can be quite noisy because many communications cannot be considered representing socio-technical relationships (e.g. not work-related).

In this work, we used only the version control change logs to create our developer network. Our developer network is a graph where the vertices represent a developer on the team. Edges exist where two developers made a version control commit to the same source code file within one month of each other. An analysis of our choice of one month for edges can be found in Section 6.4.

Additionally, a single edge between two developers might seem too coarse of a measure for measuring socio-technical distance. Without weights, two developers who only worked on only a file or two by would be treated the same way as two developers who worked on dozens of files together. To provide a finer granularity in distance measurement, we evaluate a weight to the edges in our developer network.

When applying edge weight techniques, the weight must adhere to an interval scale [5] so that addition can still be used when computing geodesic paths. In the case of using file counts for an edge weight, we must also make sure that weight represents *distance*, thus more files must represent a smaller distance.[2]

The weights of our edges are defined as the number of source code files in the system that the developers did *not* work on together. For example, if there are 1000 files in a system, and two developers worked on 10 different source code files together, then the weight on the edge (representing distance) would be 990. In this study, we evaluate both weighted and unweighted edges separately.

[2] Applying inversion to the number of files (i.e. 1/N) does not preserve interval scale.

An example of version control change logs and resulting developer network can be found in Table 1 and Figure 1 respectively. Assume that the system consists of ten files in total, and that all of the following changes are within the same month.

Table 1. Example version control logs.

Developer	Changed Files
Alex	A, F, G, H
Ichiro	E
Ken	A, B, C, E
Randy	B, C, D, F, G, H

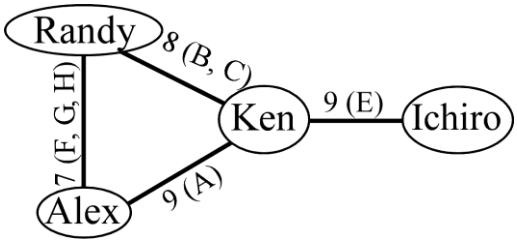

Figure 1. Example developer network.

5. RESEARCH METHODOLOGY

In this section, we discuss our online survey and the steps we took to conduct this study.

5.1 Online Survey

Our online survey consisted of six questions posed to the developers. The entire survey can be found in the Appendix.

For each question, the respondents were given the opportunity to provide feedback on the questions itself. On some of the questions, the developers clarified how they interpreted the question (e.g. the meaning of "work with" described in Question 2). We found that some questions were interpreted differently among the developers. Based on that feedback, we only analyze three of the six questions in the survey.

Our online survey is personalized for each developer of a given project. The possible answers for each developer is slightly different based on the SNA metrics of the respondent.

The developers of each development team were each sent a solicitation email for the survey. Each email had a personalized link to the survey. Before the survey begins, the respondent must first verify his or her own identity. After verifying his or her identity, each respondent answered a series of questions. The specific questions used in this analysis are discussed further in Section 6.

Our survey system was developed in Java, JSP, and JavaScript, running in Tomcat 5.5, MySQL 5, on a server running Red Hat Enterprise Linux 5.

5.2 Conducting the study

We executed the following steps to perform the study.

Step 1: Obtain version control data. For each of the case studies, we obtained the version control data from the project websites. In each case study, the version control data was captured from July 1st 2008 through July 1st 2010.

Step 2: Compute the developer network. We computed our developer network according to the definitions in Section 2 using a combination of our own Java scripts, the JUNG[3] framework, and the MySQL relational database.

Step 3: Calculate geodesic path distances. Our geodesic path distance measures were calculated with both weighted and unweighted edges. The weights of our edges are defined as the number of source code files in the system that the developers did *not* work on together (see Section 2 for an example).

Step 4: Obtain emails for each developer. We obtained developer email addresses from public artifacts, including the version control system, mailing lists, and issue tracking databases. In all three studies, we were able to trace every version control ID to an email, although the process was manual.

Step 5: Develop online survey. See Section 4.1 for a description of our survey.

Step 6: Load geodesic path distances from the developer network into the survey. The formulation of some questions depended gathering developers of varying socio-technical distances from the respondent. We used the weighted geodesic path lengths from the developer network (defined in Section 4) as our measure in the survey.

Step 7: Solicit developers to take the survey. We sent out a solicitation email to every developer in each developer network, asking them to follow the link to our survey. The link contained a key that was specific to each developer so that the survey system could track the respondent's answers. As an incentive, we put each developer in a drawing for a gift certificate to Amazon.com.

Step 8: Process and analyze responses. Before analyzing the data, we read all of the comments that people left on each question and processed the data. In some situations, we removed some responses as being non-participatory in the study (those responses are not counted in this paper). We also removed questions from our analysis based on developer feedback.

The specific details of the case studies can be found in Section 5, and the results of the analysis can be found in Section 6.

6. CASE STUDIES

In this section, we describe the details of the three case studies we performed. We intentionally chose projects from different domains with varying community sizes.

In all three case studies, when we refer to "developers", we are referring to the core group of people who are actively working on the project. In most open source development, these core developers are called "committers" because they have access to making commits to the development project.

Additionally, in all three case studies we only included commits to source code, which we defined as files ending in .c, .cpp, .S, and .h. We used the previous two years of development history, ranging from July 2008 to July 2010.

Table 2 shows meta-data on our three developer networks, along with the response rates to the survey.

[3] http://jung.sourceforge.net

Table 2. Case study meta-data

	Linux kernel	PHP	Wireshark
# Developers	226	76	28
# Edges	827	492	203
# Commits	46,201	7,975	10,659
Network diameter	9	4	3
# Respondents (%)	90 (40%)	18 (24%)	16 (57%)

6.1 Linux kernel

We performed our case study on the most recent version of the Linux kernel at the time of this writing (Version 2.6.35). For the version control data from which developer activity metrics were computed, we used the Linux kernel Git source control repository[4].

In the Linux kernel, the version control repository makes an explicit distinction between the "committer" of change, and the "author" of a change. For example, if Alison submits a patch, but Rob commits the change, then Alison is credited as the author, and Rob is the committer. We used the "author" field in this study because the author is intended to be the person directly involved in the change to the system.

The Linux kernel had over 4,000 authors in the time period we examined. The vast majority of these people, however, made very small changes to the system. Since we are focusing on the core group of developers in this study, we only included the 226 developers who were specified as authors of at least 50 commits in the past two years. Of all the 46,201 commits in the previous two years, the 226 authors were on 67% (31,081) of the total commits. We obtained all 226 developer emails directly from the version control system.

Figure 2 depicts the Linux kernel developer network. The high density of edges highlights the large volume of development activity being done by multiple developers. Also, the entire network is spread out with a single group of central developers in the center. The wide diameter of the network (9 as shown in Table 2), supports the visual result that the network is spread out.

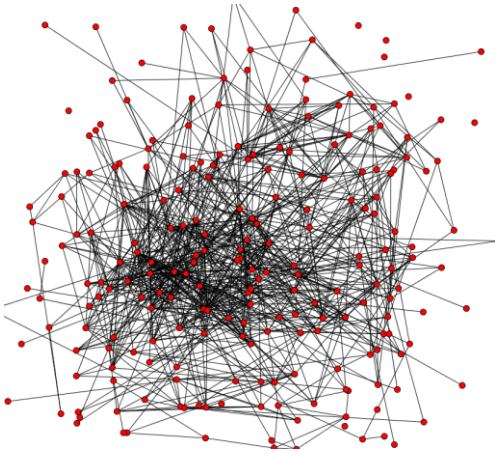

Figure 2. Visualization of the Linux kernel developer network.

[4] http://www.kernel.org

6.2 PHP Programming language

We performed a case study of the PHP programming language. The PHP programming language project includes development on the language itself, as well as extensions and modules. For this study, we focused on the developers working on the language itself. Unlike the Linux kernel, the main group of developers is more precisely defined as the people who are granted commit access to the Subversion repository. Thus, we focused our study on the group of committers to the PHP language itself.

We obtained the 76 email addresses for each developer by manually investigating their version control user ID in public sources such as the profiles and mailing list archives on PHP.net[5].

Figure 3 depicts the developer network for the PHP programming language. As with the Linux kernel, the central developers within this team are quite apparent in this layout. However, the overall spread of the network is much tighter, supported by the diameter metric being 4.

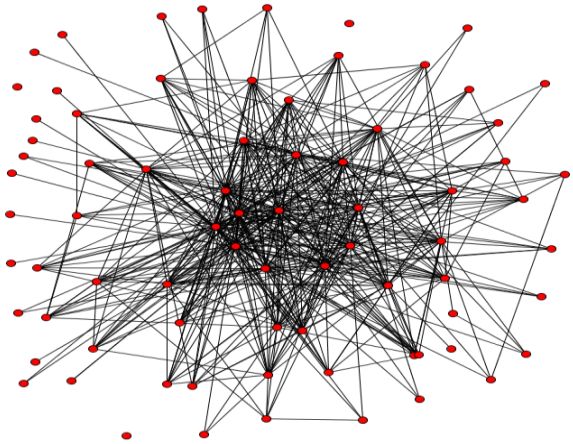

Figure 3. Visualization of the PHP developer network.

6.3 Wireshark network protocol analyzer

The Wireshark network protocol analyzer is a tool that can be used to aggregate and summarize data transported over a network. We focused our study on the group of committers using their Subversion repository.

We obtained the 28 email addresses for each developer by manually investigating their version control user ID from their public website[6].

Figure 4 shows a visualization of the developer network for Wireshark. The central developers are not as easily seen, mostly because of the high degree of most developers. Additionally, this community is much smaller (28 committers), so the spread of the network is much smaller, as supported by having a diameter of 3.

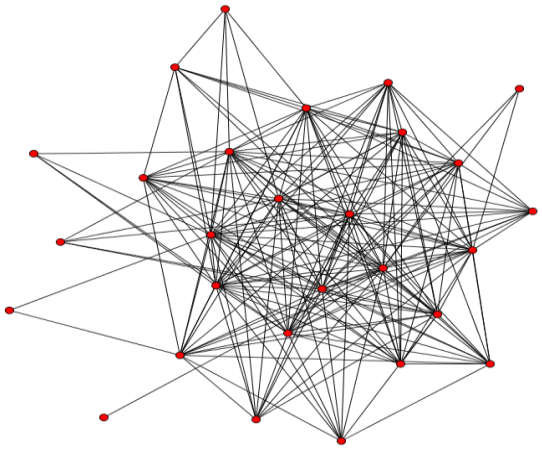

Figure 4. Visualization of the Wireshark developer network.

6.4 Criterion for Developer Network Edges

As we stated in Section 4, for our developer networks we defined edges where two developers made version control commits to the same source code files within the same month. That one month time period, which we will refer to as the *edge window*, is a parameter that can be tweaked for different software projects depending the process and team culture.

Other studies [4,15,18] implicitly use an edge window of infinity, thus defining an edge where two developers change the same source code file at any time in history. We believe that such a definition did not incorporate the ephemeral nature of collaboration: as code changes, so do the socio-technical connections amongst the teammates. With an infinite edge window, a developer could potentially be connected to other developers for years after the code has completely changed. Thus, we sought to find a minimal, meaningful edge window according to the collaboration we observed in our case studies.

We arrived at our choice of the one month edge window through a manual investigation of the frequency of regular committers to features in all three case studies. Through reading version control logs, mailing lists, and issue tracking data, we identified regular committers to each project. We observed that even the most frequent committers to a project would sometimes drop off in commits for multiple weeks at a time. The main reasons we could glean from the artifacts included: temporarily working on other projects, vacation, and the product undergoing stabilization prior to release. Based on these observations, we decided that a 30-day edge window was a reasonable choice.

To understand the impact of the edge window parameter on number of developer edges, we plot edge window values in days against the number of developer edges in Figure 5.

[5] http://www.php.net/mailing-lists.php

[6] http://wiki.wireshark.org/Developers

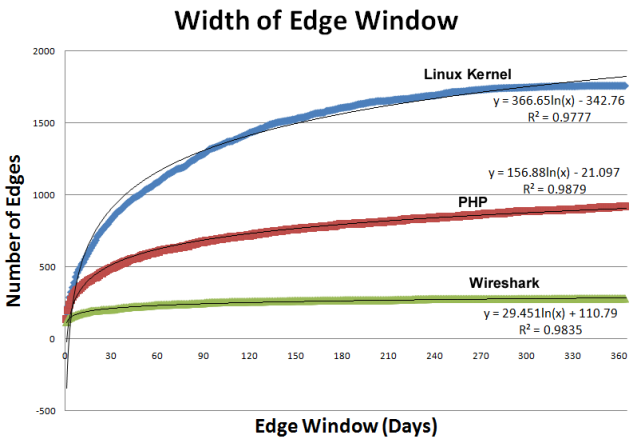

Figure 5. Width of the edge window and number of edges

We observed that the number of edges grows rapidly for 0-60 days, then slows down as the window widens out to a year. Interestingly, we found a strong logarithmic relationship (R^2>97% in all three case studies) between the width of the window and the number of edges in the developer network. These results indicate that, in our three case studies, the growth in the number of edges changes little when the edge window parameter becomes several months wide.

While we observed one month to be a reasonable edge window for our case studies, we do not believe that one month ought to be unilaterally applied to all developer networks. Developers are working on all three of our case studies commit code constantly, which cannot be said about all software projects. We believe that the edge window parameter ought to be individually determined for each project.

7. ANALYSIS

Our main goal of analyzing this data was to examine if developer networks can be corroborated with developer perceptions. We examined specific research questions that were motivated by potential limitations of related work. The three questions that are covered relate to developer network edges (section 7.1), developer network distance (section 7.2), and developer network centrality (section 7.3).

7.1 Collaborators and Edges

Several studies employing developer networks have focused on the notion of collaboration (including our own studies), claiming that the developer network is an estimation of collaboration. The reasoning is that if two developers are working on the same source code around the same time, then some type of socio-technical relationship likely exists between the two developers, perhaps a collaboration connection.

To evaluate perceived collaboration in this study, we asked the developers who they collaborate with in the context of project. Our research question is:

*Q1: **Developer network edges**. If two developers work on the same source code file in the same month, then do they perceive they are collaborating?*

Figure 6 show the research and survey question as it was presented in our survey system. The list of names from the auto-suggest list comes from the list of committers from the version control logs.

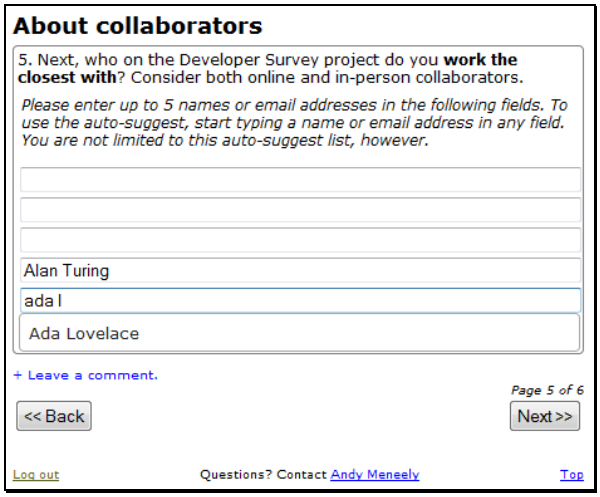

Figure 6. Screen capture of the question regarding collaborators and edges

To analyze this question, we first processed everybody's responses for misspellings of names and other input validation concerns.

Of the 460 named collaborators across all projects, 87 (19%) of those collaborators were not found in the version control change logs. We manual investigated those names and found that most people in this category were project organizers or some other role not related to development. Thus, we excluded those 87 named collaborators from our data.

Next, for the collaborators found in the developer network, we examined *unweighted* geodesic distance between each respondent and his or her named collaborators. For example, if a respondent was directly connected to a named collaborator in the developer network, then the unweighted geodesic distance was one. We aggregated each of those unweighted geodesic distances between a respondent and the named collaborators and report the results in Table 3.

Table 3. Unweighted distances between respondents and their named collaborators

	Linux kernel	**PHP**	**Wireshark**
# Named Collaborators	241	51	50
% Collaborators with unweighted distance=1	53%	43%	84%
% Collaborators with unweighted distance=2	26%	53%	16%
% Collaborators with unweighted distance=3	12%	4%	0%
Mean unweighted distance	1.8	1.6	1.2
95% CI (normal)	(1.7, 1.9)	(1.4, 1.8)	(1.1, 1.3)

In all three case studies, at least 43% of the named collaborators are directly connected in the developer network, and at least 79% of named collaborators had an unweighted distance of one or two.

In the case of PHP, slightly more named collaborators had an unweighted distance of two than one. Furthermore, the mean unweighted distance from a developer to his or her collaborators was considerably close to one. Assuming a normal distribution of the unweighted distances, the confidence interval around the average distance is fairly tight. Thus, if a developer is not directly connected to a named collaborator, then they typically are connected to someone who is connected to that collaborator (i.e. an unweighted distance of 2).

Note also that a respondent could be connected to developers whom he or she did *not* name as a collaborator. We did not assume that a developer would be able to remember or even know who all of his or her collaborators are. Therefore, this particular question can only be used to confirm the positive identifications of collaborators, not the negatives.

Also, our analysis in section 7.2 takes into account the non-existence of named collaborators.

These results indicate that *perceived collaborators are being captured by edges* in a developer network based solely on the version control change logs. In other words, collaborators in these three open source projects were often working on the same code within one month of each other. While this result substantiates that developer edges are linked to collaborators, one can see that not every named collaborator is represented by an edge. When interpreting developer networks, then, one should not assume that every collaborator is directly connected in the developer network, although empirically the distance between collaborators tends to be less than two.

7.2 The Distance between Two Developers

Some of the most commonly-used SNA metrics in related work are centrality metrics. Specifically, betweenness (defined in Section 2.1) has been used heavily to measure a developer's centrality in a network. The concept of betweenness is based on the notion of geodesic paths: if a developer is on a geodesic path between two other developers, then that developer must be central to the network. Therefore, studying the validity of the length of a geodesic path helps us study the validity of centrality metrics as well.

To examine developer perceptions of distance, we use the following research question.

Q2: ***Developer network distance.*** *Does the distance between two developers in the developer network represent the perceived degree of separation between those two developers?*

To address this research question, we provided the following question in our survey in Figure 7. The choices were

- A: I have never heard of this person before
- B: I recognize this name, but I don't know much about them
- C: I know who this person is, but I have not worked with them directly
- D: I have worked with this person on this project

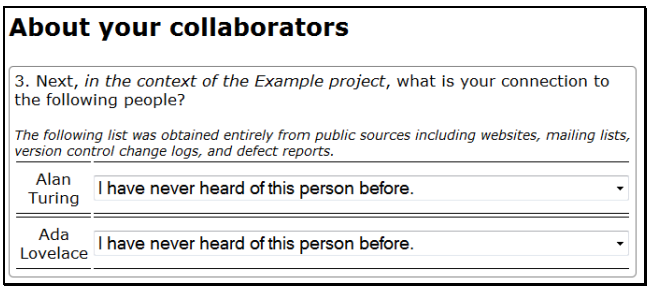

Figure 7. Example screen capture from custom survey tool

For each person, the survey showed 10 teammates from the developer network. Each respondent was given four options for each person (shown in Figure 8 as options A through D). We came up with the perceived distance levels and treat those levels on an ordinal scale with A being the most distant, D being the closest.

The survey chose those 10 teammates in the following manner. First, the system calculated the respondent's weighted geodesic distance to each of the other developers in the project's developer network. The survey then ranked all of the other developers by that weighted distance, and then chose developers of varying distances in the ranking. For example, if there were 50 other developers, then the system would rank those 50 developers by weighted geodesic distance to the respondent, and then return the developers ranked 1, 5, 10,…up to rank 50. The actual list of 10 developers was then shuffled before displaying in the survey.

Our motivation for this algorithm was this: if the geodesic distance to other developers represents the perceived distance then the ranking of developers by geodesic distance ought to match the ranking of developers by perceived distance. To measure the degree of agreement between two ranks, we use the Spearman rank correlation coefficient. The Spearman coefficient is useful when the underlying distribution of the random variables is unknown, yet they are still on the ordinal scale.

We calculated the Spearman coefficient for each respondent's answer to the question, giving us a value in the range [-1,1]. The square of the Spearman coefficient itself is normally distributed, so we report the 95% confidence intervals around the means of the squared Spearman coefficients (R^2) as well.

We report in Table 4 the mean of Spearman coefficients between the reported perceived distance from the survey and the *unweighted* distance from the developer network.

Table 4. Mean of squared Spearman rank correlation coefficients between unweighted and perceived distances

	Linux kernel	**PHP**	**Wireshark**
Mean R^2 (Spearman)	0.23	0.35	0.26
95% CI (normal)	(0.18,0.28)	(0.22, 0.48)	(0.07, 0.45)

In each case study, the mean squared Spearman coefficient was significantly greater than zero, indicating the existence of a correlation between the perceived socio-technical distances and the developer network distances. Although the correlations are statistically significant, the strength of the correlations are not

considerably close to one, indicating that the unweighted geodesic distance in a developer network does not *always* match developer perceptions.

Additionally, we applied the same analysis to the weighted distances to examine if the finer granularity in weighted edges closer match developer perceptions. Table 5 contains means of the Spearman coefficients between the reported perceived distance from the survey and the *weighted* distance from the developer network.

Table 5. Mean of squared Spearman rank correlation coefficients between weighted and perceived distances

	Linux kernel	PHP	Wireshark
Mean R^2 (Spearman)	0.23	0.47	0.32
95% CI (normal)	(0.18, 0.28)	(0.37, 0.57)	(0.18, 0.45)

The means appear to be higher for weighted edges, however, in all three case studies, the confidence intervals overlap between weighted and unweighted means, indicating that there is no statistically significant difference between using weighted developer distances and unweighted developer distances. Thus, we do not have enough evidence to claim that weighted edges are any better than unweighted edges in measuring what developers perceive as socio-technical distance.

These results indicate that the perceived socio-technical distance between two developers is statistically correlated with the geodesic distance in the developer network.

7.3 Centrality and Reputed Expertise

Now that we have shown that geodesic paths in the developer network represent developer distances, the meaning of the betweenness centrality measure becomes more intriguing. Experts in SNA in other disciplines have argued that the betweenness centrality measures represent authority and visibility in the network [5].

According to our experience, open source projects deal with visibility and authority of developers quite often. Proponents of open source development have discussed the importance of a "meritocracy" in organizing development efforts [14]. The idea is that the more a developer works on the code, the more their work will be noticed and trusted, the more authority they will have on steering the project. Therefore, in the context of open source projects, visible authority is often tied to being an expert developer on a given project.

Open source developers often measure this level of merit by counting the commits a given developer made. However, as an example, if developer Peter works on code in relative isolation from the rest of the group, then he would have a large number of commits but have no collaborations with other members of the team, and therefore a low centrality. On the other hand, if developer Linus makes many commits to files that many other people are working on, he will likely have a higher centrality.

We hypothesize that, in the context of a meritocracy, central developers are regarded as experts in the project.

Q3: ***Developer network centrality.*** *Does having a high developer centrality in the developer network indicate being a project expert?*

Our survey question regarding expertise is shown in Figure 8.

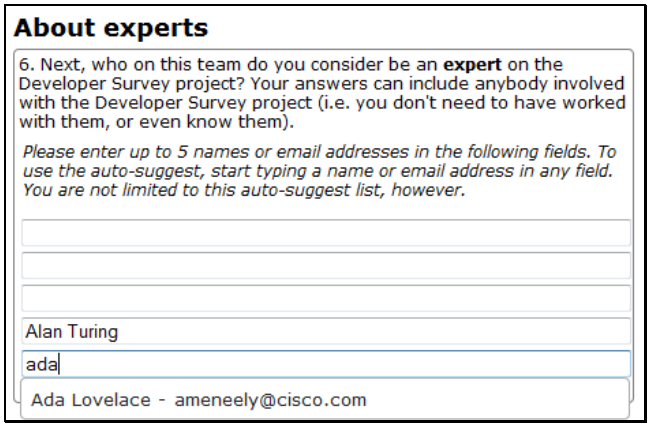

Figure 8. Screen capture of the question regarding expertise.

To analyze this question, we first processed everybody's responses to ensure that the person they named was someone in the developer network. Furthermore, we manually checked for misspellings of names.

We then counted up the number of votes for each named expert. To be considered an expert, we only considered people with three votes or more. Self-voting was allowed, and one person could not vote for another person more than once.

Some respondents named experts who were not in the developer network, which we did not include in this study. None of the 15 named experts who were not in the developer network was named more than once, so this effect did not skew our results.

Table 6 shows the breakdown of the number of experts identified and how many of the total votes that the top five experts obtained. Figure 9 shows three histograms by project of the frequency of experts (i.e. 3 or more votes).

Table 6. Summary of Votes for Experts

	Linux kernel	PHP	Wireshark
Number of votes	298	59	58
Number of experts with votes>=3	19	10	8
% of votes for the top 5 experts	84%	54%	72%

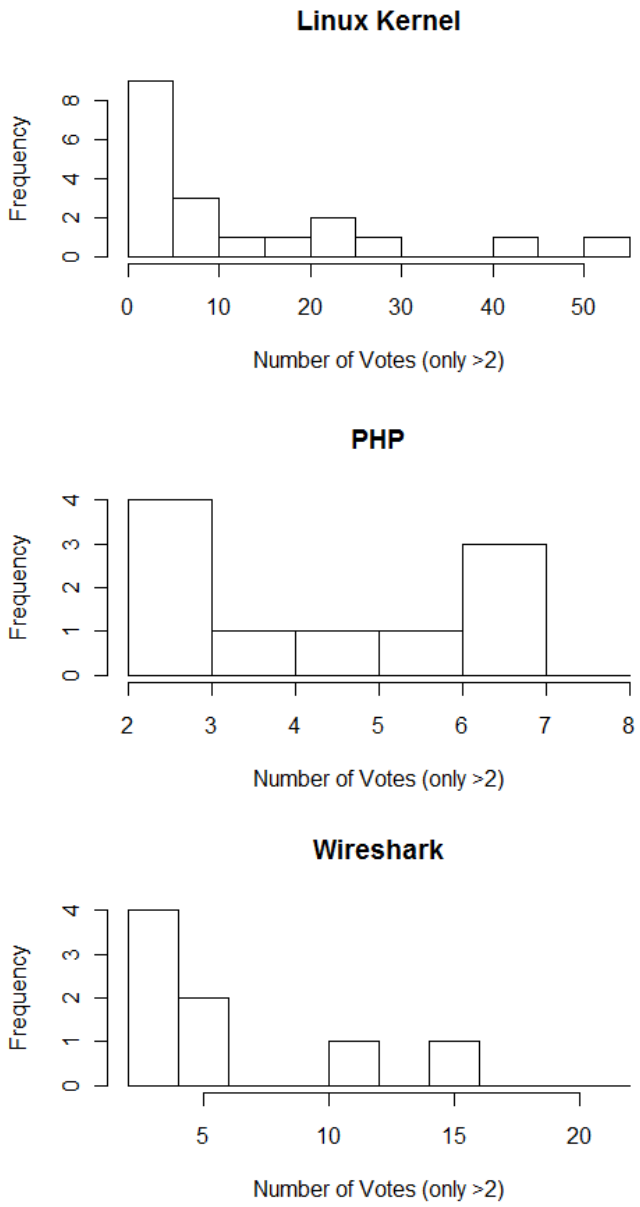

Figure 9. Frequency of project experts by number of votes.

In all three case studies, the top five experts accounted for the majority of the votes on expertise. In the case of PHP, this percentage was lower, indicating that the general regard for who was the expert was less widely agreed-upon than the other two case projects.

To examine if centrality is correlated with reputed expertise, we used the non-parametric Mann-Whitney-Wilcoxon (MWW) test. We tested for a statistically significant difference in our two centrality metrics developers regarded as experts and developers not regarded as experts. We apply our test to our two centrality metrics: degree and betweenness. Table 7 contains the results of those tests.

Table 7. Association between centrality and reputed expertise

Centrality Metric	Statistic	Linux kernel	PHP	Wireshark
Degree	**Mean for Expert**	14.0	33.6	19.9
	Mean for Non-Expert	6.2	8.8	11.4
	Difference significant ($p<0.05$)?	Yes	Yes	Yes
Between-ness	**Mean for Expert**	0.02	0.06	0.02
	Mean for Non-Expert	<0.0001	<0.0001	0.01
	Difference significant ($p<0.05$)?	Yes	Yes	Yes

The betweenness measure can be interpreted as the percentage of total geodesic paths on which a developer can be found. Being on just 2% of the total geodesic paths, for example, is quite central for a single developer. As one can see from the results, experts tend to be at least 2% of the total geodesic paths *on average.* In every case study and for both degree and betweenness, reputed experts were statistically more likely to have a higher centrality than non-experts.

Our results indicate that developer network centrality is associated with being a reputed expert in the project.

8. LIMITATIONS

The largest limitation to this work is the validity of developer perceptions. Developers can be wrong about who they collaborate with, which could influence our results positively or negatively. While perceptions are not perfectly trustworthy, we believe that comparing developer perceptions with the artifacts brings us closer to better measuring the overall structure of collaboration in a software development team.

Not every developer on the project responded to the survey solicitation, and we do not know if our sample of developers was biased in some way. To mitigate this, we examined three different projects in an effort to achieve more generality.

Our questions regarding experts and collaborators required the user to remember people's names. A respondent might not be able to recall the name of an expert or collaborator. To mitigate this, we provided an auto-suggest list of the committers. Additionally, in our analysis, we did not assume that the named collaborators are a *complete* list of collaborators.

Lastly, since our approach included only committers of a project, our results do not generalize to the structure of non-coding participants surrounding a software development project. Such participants can include users, managers, and non-coding support. Our reasoning for this particular scope was based on the information found in the artifacts we chose to study.

9. CONCLUSION

The objective of this research is to evaluate the validity of SNA metrics by corroborating developer networks with the perceptions

of developers of the projects involved. Our results indicate more support for researchers' assumptions on what developer networks represent. In each case, however, not every notion was fully supported by developer responses. For example, while edges are close linked to collaborations, not every edge represents collaboration and not every collaboration is represented by an edge. Social network analysis techniques that are sensitive to missing edges, then, are not appropriate to developer networks. Overall, the developer network ought to be treated as an approximation in future applications of social network analysis. Nevertheless, the developer network is, in general, supported by developer perceptions. Our results can help researchers and practitioners understand the complex ecosystem of developer activity on software development teams.

10. ACKNOWLEDGMENTS

We thank all of the developers who responded to our survey. We also thank the reviewers, the NCSU Realsearch group, and Pete Rotella from Cisco for providing valuable feedback on the paper. This research is supported by the Army Research Office.

11. REFERENCES

[1] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: Discovering and Exploiting Relationships in Software Repositories," in Int'l Conference on Software Engineering (ICSE), Cape Town, South Africa, 2010, p. 125-134.

[2] C. Bird, A. Gourley, P. Devanbu *et al.*, "Mining Email Social Networks in Postgres," in Mining Software Repositories, Shanghai, China, 2006, p. 185-186.

[3] C. Bird, N. Nagappan, P. Devanbu *et al.*, "Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista," Comm. of the ACM, vol. 52, no. 8, p. 85-93, 2009.

[4] C. Bird, D. Pattison, R. D'Souza *et al.*, "Latent Social Structures in Open Source Projects," in FSE, Atlanta, GA, 2008, p. p24-36.

[5] U. Brandes, and T. Erlebach, *Network Analysis: Methodological Foundations*, Berlin: Springer, 2005.

[6] T. G. Cummings, "Self-Regulating Work Groups: A Socio-Technical Synthesis " The Academy of Management Review, vol. 3, no. 3, p. 625-634, 1978.

[7] A. Meneely, M. Corcoran, and L. Williams, "Improving Developer Activity Metrics with Issue Tracking Annotations," in Workshop on Emerging Trends in Software Metrics (WETSoM), Cape Town, South Africa, 2010, p.

[8] A. Meneely, and L. Williams, "Secure Open Source Collaboration: An Empirical Study of Linus' Law " in Computer and Communications Security, Chicago, IL, 2009, p. 453-462.

[9] A. Meneely, and L. Williams, "Strengthening the Empirical Analysis between Developer Collaboration and Software Security," in Empirical Software Engineering & Measurement (ESEM), Bolzano-Bozen, Italy, 2010, p. to appear.

[10] A. Meneely, L. Williams, J. Osborne *et al.*, "Predicting Failures with Developer Networks and Social Network Analysis " in Foundations in Software Engineering, Atlanta, GA, 2008, p. 13-23.

[11] N. Nagappan, B. Murphy, and V. R. Basili, "The Influence of Organizational Structure on Software Quality," in International Conference on Software Engineering, Leipzig, Germany, 2008, p. 521-530.

[12] R. Nia, C. Bird, P. Devanbu *et al.*, "Validity of Network Analyses in Open Source Projects," in Mining Software Repositories, Cape Town, South Africa, 2010, p.

[13] M. Pinzger, N. Nagappan, and B. Murphy, "Can Developer-Module Networks Predict Failures?," in Foundations in Software Engineering, Atlanta, GA, 2008, p. 2-12.

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

[15] A. Sarma, L. Maccherone, P. Wagstrom *et al.*, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, 2009, pp. 22-33.

[16] Y. Shin, A. Meneely, L. Williams *et al.*, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," IEEE Transactions in Software Engineering (TSE), vol. to appear, no. p. 2010.

[17] E. Trist, and K. Bamforth, "Some social and psychological consequences of the longwall method of coal getting," Human Relations, vol. 4, no. 1, p. 3-38, 1951.

[18] T. Wolf, A. Schroter, D. Damian *et al.*, "Predicting build failures using social network analysis on developer communication," in Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, 2009.

12. APPENDIX

The removed survey questions are provided here. Based on respondent feedback, we do not present or analyze the results of questions one, two, and four. Additionally, wherever [project] appears, the survey would replace with the name of the project (e.g. Linux kernel project). A description of the type of input the survey allowed is in italics.

Removed Question 1. On the [project], I perform the following tasks (check all that apply): Write code, Write and/or execute tests, Design software, Manage people, Inspect other peoples' code, Fix defects, Answer questions from customers, Answer technical questions from fellow developers, Steer the overall direction of the project.

Removed Question 2. In your estimation, how many different members of this project have you worked with in the last month on the Developer Survey project? Include in your count both in-person and online interactions. Do not include yourself in this count. *The user can enter a positive number for this question.*

Removed Question 3. Consider the following scenario. Suppose you are developing a new feature for the Developer Survey project, and you realized that your changes could make the system **insecure** if your implementation is not correct. You decide to contact some of your colleagues to inspect your feature to ensure that no security vulnerabilities are being introduced. What factors are most important to you in deciding **who to work with** in this situation? Please assign 24 points to each of the following 8 factors, giving a higher weight for a higher importance.

- I work with this person frequently.
- This person is conveniently located near me.
- This person knows a lot about software security in general.
- This person has worked on similar features to this one before.
- In the past, this person has worked on parts of this project with high security risk.
- This person is highly experienced in software engineering.
- Someone I respect recommended this person.
- This person is my superior.