Network Analysis of Software Repositories: Identifying Subject Matter Experts

Andrew Dittrich, Mehmet Hadi Gunes and Sergiu Dascalu

Abstract A software developer joining a large software project faces a steep learning curve before they are able to make real contributions. One challenge is finding the subject matter experts who can answer questions about a specific area of the software or to review changes. This is especially true of large projects with many modules and a large number of authors. In this paper, we describe a method to model a software project as a network using information mined from the project's version control repository, and demonstrate how network analysis techniques can be used to identify the key authors and subject matter experts. We investigate metrics that can be gathered using network analysis, such as which groups of authors typically work together, and how closely knit the developers are on a project. We analyze several specific projects to demonstrate the applicability of these techniques and several hundred projects to show general trends.

1 Introduction

A new developer starting on a large has a lot to learn before they can be a productive member of the team. The project contains many different modules, each of which can be complex on its own. Typically, a junior developer will turn to a more senior developer to ask questions, and to gain insight into the overall architecture of a project. However, it can be difficult to identify experts for a particular area. A good candidate to start with is the person who last modified a file in a module, but this

Andrew Dittrich
University of Nevada, Reno, e-mail: andy.dittrich@gmail.com

Mehmet Hadi Gunes
University of Nevada, Reno, e-mail: mgunes@cse.unr.edu

Sergiu Dascalu
University of Nevada, Reno, e-mail: dascalus@cse.unr.edu

person may have just fixed a formatting problem or a compiler warning, and might not be the best person to ask.

Identifying the most experienced author for a specific area of the project is also a problem for project managers. If a bug is found in a specific module of a large software project, then ideally, the most experienced developer in that area of the project should be assigned to fix it. Unfortunately, there is not an easy way to identify that individual. If the manager has been working on this project for a while, then they most likely have the experience to know who the key developer is in this area. Alternatively, they can survey the team members to find someone who is familiar with the area of the code in question.

A project manager may also be interested in how the development team works together. If each developer works on a separate part of the project, and there is no overlap in responsibilities, then there is increased organizational risk from team members leaving the organization. A manager can mitigate this risk by analyzing which members work together and organizing the team such that there is more overlapping knowledge [12]. This risk is difficult to quantify, as there are limited methods for measuring team cohesiveness.

Researchers have investigated collaborative networks to understand different aspects of collaborations [8]. This paper proposes modeling the version control repository as a network, and applying network analysis techniques to identify the key authors for the project and to measure team cohesiveness.

The next section discusses related work. Section 3 discusses how data can be gathered from a source control repository. Section 4 discusses how network analysis techniques are applied. Sections 5 and 6 discuss the results of this analysis on some specific projects, and general trends resulting from the analysis of a few hundred projects. Section 7 analyzes the results. Section 8 concludes the article and suggests future research in this area.

2 Related Work

There are many metrics that can be used to analyze a software project, but there are very few metrics to identify key authors. Commonly used metrics include defect rate, complexity, test coverage, and productivity [10]. These metrics are rarely used to judge a specific author. Associating software metrics with specific authors can cause authors to feel threatened, and is not recognized as a best practice in industry [13]. Hence, typical software metrics are not available to solve this problem.

Other techniques have been developed to identify individuals familiar with specific areas of software. One such method is described by Linstead et.al. [6]. This method searches the source code for keywords or topics, and associates authors with the topics based on the history contained in the revision control repository. This method is able to identify an author who is familiar with a particular topic in the source code. Based on their results, this method is effective in identifying subject matter experts for specific areas of code. However, this method does not

consider which authors are the core developers for the overall project, and does not take advantage of the existing relationships between authors that are available in the version control repository.

Another network analysis method is described by Lopez-Fernandez et.al. [7]. This method mined open source version control repositories to identify networks of authors and gain insight into the overall structure of a group of developers. The approach connects two authors if they have contributed to the same module, and produces an author network. General data is then gathered from this network in order to characterize the project overall.

Huang et.al. describe a similar analysis technique where an author network is created using data from a source control repository [3]. Authors are connected if they have worked on a file in the same directory. The resulting author graph is analyzed using distance centrality to separate the network into kernel and peripheral developers.

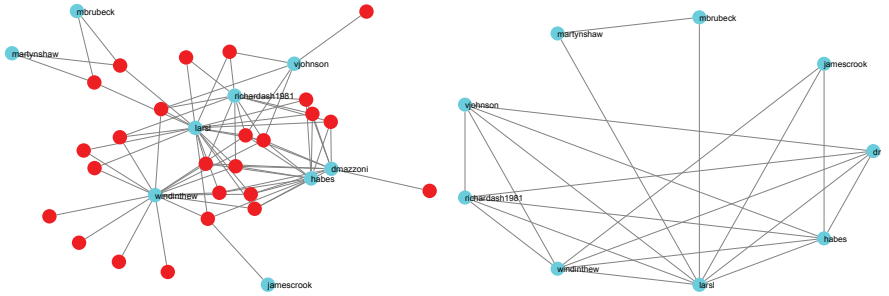
Several articles have been written on methods for gathering data from source control repositories. Voinea et.al. describe a framework for querying CVS repositories, parsing the data, and analyzing it. They also propose a method to visualize the resulting data to highlight patterns in the development of a project, such as changes in the development team over time [14] [15]. Kagdi et.al. describe a method to recover the ordered sequence of changed items in a Subversion repository using several heuristics, and a method for analyzing the results [5]. These advanced repository mining techniques were not required for this study. The data needed from a repository can be easily obtained from a log file and converted into a graph for further analysis, as described in the next section.

Several studies have been performed to identify connections between members of networks. Extensive analysis of authors of academic papers has been performed to identify relationships between authors and author groups [9]. This analysis takes a similar approach by linking authors that worked on the same paper, and using this information to create an author network. This differs slightly from the software collaboration. Coauthorship of an academic paper consists of multiple authors working together at the same time, whereas two developers may work on the same source code at separate times with less collaboration between authors.

3 Gathering the Data

The first step in this analysis is to create a bipartite graph that links each unique author with the files that they changed. One set of vertices in the graph are authors and the other set are files. An edge is created for each author that changes a file. For a basic analysis, each edge has equal weighting.

The primary source for this data is a log file produced from the version control system. In this case, projects using Subversion were analyzed, and an xml-format log was produced containing details on every modification to every file in the repos-



(a) (b)

Fig. 1 Graph representations of the Audacity project. (a) A portion of the bipartite graph representing the Audacity project. Authors are blue and files are red. (b) The author graph resulting from the projection of the bipartite graph.

itory. Other projects using other version control systems such as Mercurial, Git, CVS, or Perforce could have also been used.

The log data was analyzed to produce a list of author-file pairs for each author that made a change to each file. This was interpreted as an edge list for a graph that represents the repository, i.e., a bipartite graph where one type of vertex represents a file and another type of vertex represents an author. A subsection of the bipartite graph produced for the open source Audacity project is shown in Figure 1a.

In some cases, only a subset of the repository needs to be analyzed. To accomplish this, the input data can be filtered to get more specific results using several methods. This can be done by analyzing a log for only one section of a repository, e.g., a single folder or module. The data could also be filtered by file name filters, e.g., '*.cpp'.

Another problem is with the initial addition of files. The author that adds a file will be associated with that file, even if they are not an expert in that area. To avoid this, the initial addition of files can be ignored, and only modifications are considered in the analysis.

Filtering the data by the date may also be necessary for long-term projects. Over many years of development, authors will tend to make connections to each other by working on the same file. This may give the incorrect impression that a development team works closely together while certain individuals might have never met. Filtering the data by a specific time period will avoid this problem.

4 Analyzing the Data

The graph that was produced by analyzing the data is a bipartite graph with edges between authors and the files that they modified. This can be projected into two undirected one-mode graphs that show the relationships between authors and the

relationships between files separately. The one-mode author graph produced from this projection has a vertex for each author, and an edge connects two authors if they made changes to the same file. This one-mode graph represents the network of connections between authors. For instance, the author graph resulting from the projection of the graph in Figure 1a is shown in Figure 1b.

The core developers for a project can be identified by analyzing the author graph. These are the authors that are the most connected in the author graph. If an author is well connected, then it indicates that they have worked on many different files with many different authors, and most likely have a wide range of knowledge in the area.

To measure how well connected an author is, we can check the centrality of the author. Measuring the degree of the author is a straightforward way to measure. However, this ignores the degree of the other authors to which this author is connected. If an author has connections to others with many connections, then this can indicate that the author works with other important authors, and should have a higher weight. Another measure of centrality that takes this into consideration is the eigenvector centrality. Measuring the eigenvector centrality for an author is a good indication of how well connected this author is in the author network, and can be used as a proxy to find the experts in this area.

It is important to know which authors typically work together on a software project. This is very useful for a project manager when assigning resources to a specific project. Authors who have a history of working well together tend to make a more productive team than those who don't. Hence, identifying these authors might be beneficial. One way to do this is to identify the communities of authors in the author graph. There are several algorithms for doing this. The algorithms that gave the best results in our analysis were the greedy method, the modularity maximization method, and the spinglass method.

The greedy method is a very simple algorithm that runs in $O(n\log^2 n)$ time, and is well suited for extremely large networks [9]. This algorithm is implemented in iGraph's community_fastgreedy method [4]. The projects analyzed in this study had between 4 and 158 authors, so the simplicity of the greedy algorithm was not necessary, and more complex algorithms could be explored.

The modularity maximization method is discussed in Newman [9]. This method breaks the network into communities such that the total modularity of the network is maximized. This algorithm is also implemented in iGraph's community-leading-eigenvector method [4]. This algorithm resulted in many small communities. Hence, it may be useful if identifying small teams of programmers to work together or for pair programming.

The spinglass method is a complex algorithm that simulates the cooling of a hot system into a grounded state [9]. It associates negative modularity with the energy of an infinite range spin glass and attempts to minimize the energy of the system to find communities [11]. This algorithm is implemented in iGraph's community-spinglass method, as well [4]. This method produced a few large communities in the projects analyzed, and seemed to give the best results among all methods.

The communities determined by either of the methods can be used to set up the optimal team structure for a project by selecting people with a collaboration history

Table 1 Top 10 authors as measured by centrality metric

Audacity		Subversion		Super TuxKart	
1.000	richardash1981	1.000	cmpilato	1.000	cosmosninja
0.971	dmazzoni	0.998	maxb	0.976	hikerstk
0.969	llucius	0.997	kfogel	0.955	auria
0.968	vjohnson	0.995	hwright	0.924	mbjornstk
0.964	jamescrook	0.995	dlr	0.896	coz
0.964	msmeyer	0.994	blair	0.880	hiker
0.954	mchinen	0.992	julianfoad	0.805	grumbel
0.953	windinthew	0.991	brane	0.791	thebohemian
0.951	martynshaw	0.991	ehu	0.791	scifly
0.947	mbrubeck	0.989	sussman	0.744	donconso

for new projects. Alternatively, a project manager could pick people from different communities to encourage cross-team cooperation.

The author graph can be analyzed to determine how the authors work together. Ideally, each author would be connected to each other author. This would indicate that every author had worked together with every other author, and there are at least two authors familiar with every file. So the risk of losing a key employee would be mitigated because there is always a backup who is familiar with the code.

This can be measured by the transitivity or clustering of the author graph. A high clustering coefficient indicates that many authors are connected, and a low clustering coefficient indicates that authors typically work alone.

5 Results for Specific Projects

The analysis methods described above were applied to three specific open source projects, namely, Audacity, Subversion, and Super TuxKart.

Audacity is an open source audio editing program. The Audacity project was first hosted on SourceForge in May of 2000, and has 60 unique authors, 9450 unique files, and 24,377 modifications connecting them. The core developers identified by the analysis techniques described above are shown in Table 1. The results were confirmed based on developer credits available on the Audacity website, and indicates that this technique can identify the core developers of the project.

Communities of developers in the Audacity project were identified using the spinglass technique as in Figure 2. It is difficult to verify that these communities are accurate without knowledge of the developers or experience working on this project.

The Subversion project started using Subversion for source control (self-hosting) in August of 2001 [1], so there is an extensive history consisting of 158 unique authors, 6752 unique files, and 92,775 modifications connecting them. The core developers identified are shown in Table 1. Again, these results were confirmed using information available on the Subversion website.

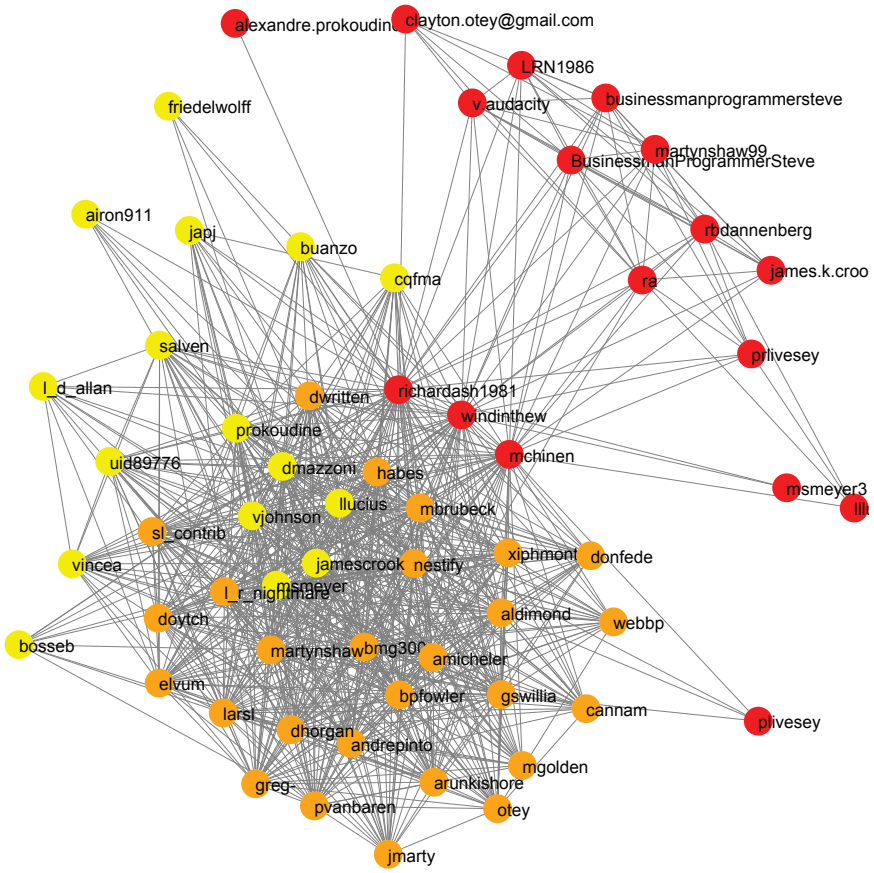


Fig. 2 Communities of authors in the Audacity project

Three communities of developers were identified using the spinglass algorithm to analyze the entire subversion project. Due to the lengthy history of the project, the results are difficult to interpret. Over such a long time, it is likely that many developers would develop connections to many others. For example, if a single file has a history of 20 revisions, then an author could potentially make 20 connections when this file is changed. This leads to a highly connected author graph without clearly defined communities.

Super TuxKart is an open source multiplayer racing game. It was based on TuxKart, and became Super TuxKart in 2006. The project consists of 36 unique authors and 12,597 unique files. The core developers identified by the analysis techniques described above are shown in Table 1. These results were verified based on the list of core developers listed on the Super TuxKart website, as well.

6 General Results

This analysis technique was automated and applied to a large number of projects. These results can be used as a guideline to see how a specific project's metrics compare to projects in general.

Open source projects from SourceForge were analyzed in bulk. The top 10,000 projects as measured by weekly downloads were gathered. Of these 10,000 projects, 5,031 used a Subversion repository. 1,063 of these repositories had no data available and were discarded. Many of the remaining projects had only a few authors, which would not give meaningful results. In order to get meaningful data about the relationships between authors, a minimum of 15 authors was chosen. This eliminated 3,665 projects, which left 303 projects for this analysis. This included projects with up to 158 authors and between 377 and 192,121 files.

Each project was analyzed to identify the core developers, author communities, and clustering coefficient for the author graph. The clustering coefficient was of the most interest. The distribution of the clustering coefficient for the author graphs in Figure 3 reveals that most projects have a clustering between 0.7 and 0.9. Audacity had a clustering of 0.783, Subversion had a clustering of 0.880, and Super TuxKart had a clustering of 0.626.

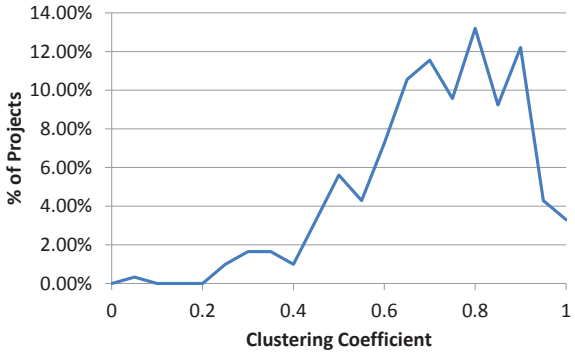
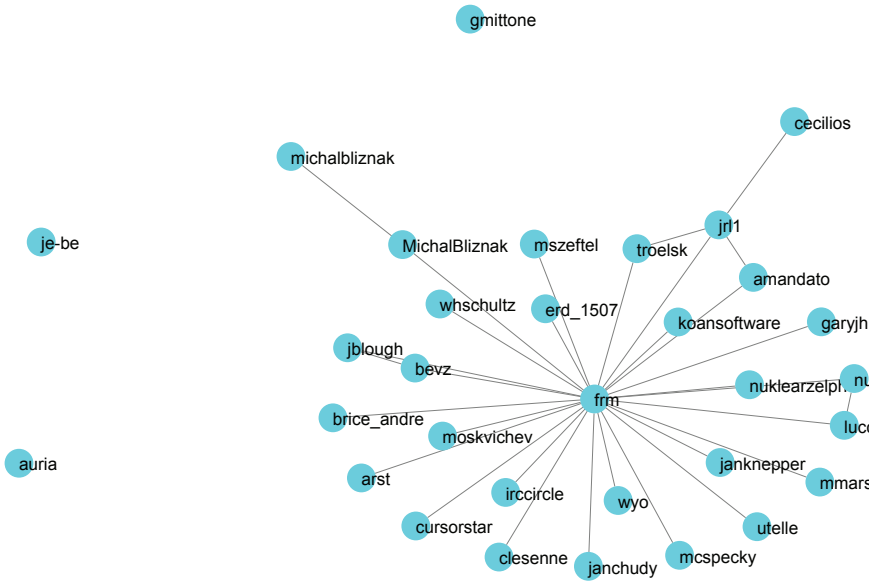


Fig. 3 Clustering coefficient distribution of all projects

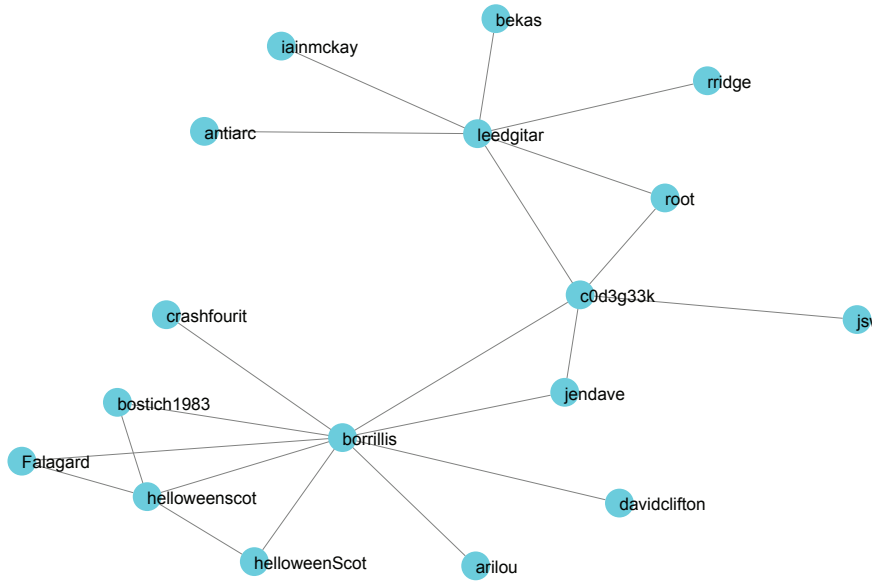
The project with the lowest clustering was wxCode with a clustering of 0.036 and its author graph is shown in Figure 4a. This project is a collection of add-on components and libraries for use with wxWidgets. Each component is separately maintained by a different author, which explains why the authors in this project typically don't work together.

The project with the second lowest clustering was Axiom 3D Engine with a clustering of 0.208 and its author graph is shown in Figure 4b. This project is a cross platform 3D rendering engine, and has 17 authors for 20,890 files. The low clustering coefficient indicates that the authors typically don't work together, which makes sense considering that there are a few authors and many files.

The project with the highest clustering was pkgbuild with a clustering of 0.991 and its author graph is in Figure 5a. This project is a tool for building Solaris SVr4 or IPS packages, and has 70 unique authors for 4,145 unique files. Another example of a project with a high clustering coefficient is MegaMek with a clustering of



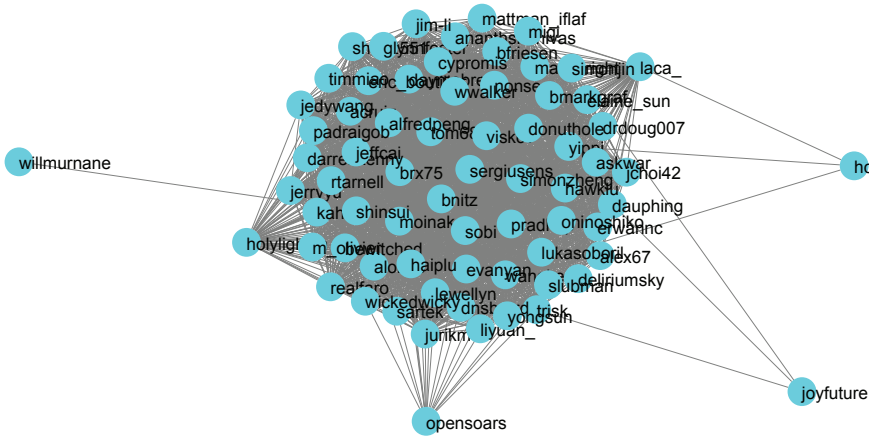
(a) wxCode



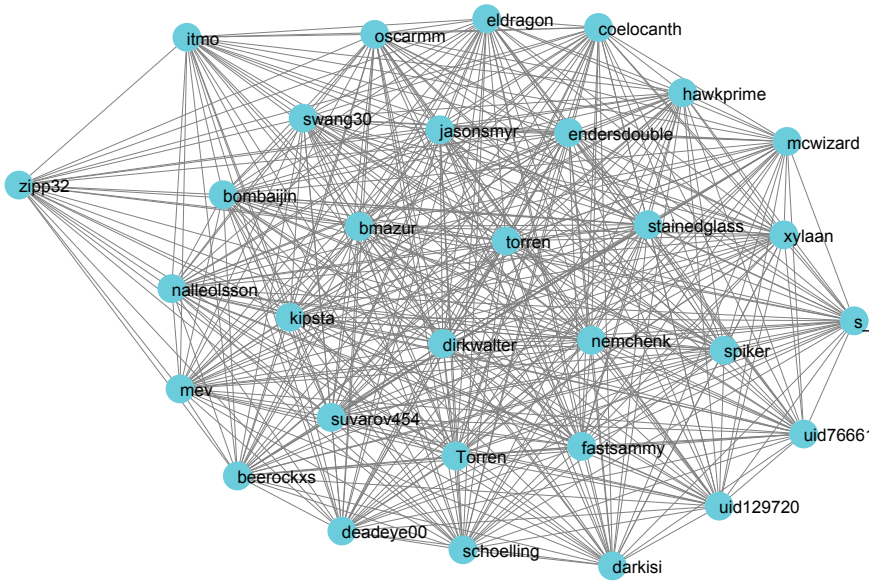
(b) Axiom 3D Engine

Fig. 4 Author graphs with the lowest clustering coefficient. (a) wxCode project with a clustering of 0.036. (b) Axiom 3D Engine project with a clustering of 0.208.

0.980 and its author graph is in Figure 5b. This project is an online version of the BattleTech board game, and has 31 unique authors for 10,735 unique files.



(a) pkgbuild



(b) MegaMek

Fig. 5 Author graphs with the highest clustering coefficient. (a) pkgbuild project with a clustering of 0.991. (c) MegaMek project with a clustering of 0.980

7 Analysis

There are several things that can impact the results from this analysis. Our method assumes that the changes made by each author are relevant to the file being modified. This is not always true. An author could make a change to the formatting of a file

or correct a typo in a comment. This author would then be linked to that file and all of the authors who had modified it previously. This should not be a common occurrence, but it has the potential to affect the results of the analysis and make some authors seem to be core developers when they really are not.

A developer's personal software practice could also skew the results. If an author makes many small changes to a file, then their connection to that file will have a higher weight than an author who makes one large change. If edge weights are ignored, then this is not a problem, but that could also skew the results towards authors who made a single change.

Another potential problem is anonymous contributions or multiple online identities for the same person. The projects analyzed in this study did not allow anonymous users to change the source code. If a project did allow anonymous users, then there would be a disproportionate number of changes associated with the anonymous user, and this anonymous user could appear to be a core developer, even though it represents many unique individuals in reality. To protect against this, each anonymous user should be considered as a separate developer [2]. Similarly, one person may have several different online identities that are used to make changes, which could prevent this person from being identified as a core developer, or even put that person in multiple developer communities.

8 Conclusion and Future Work

It can be difficult to identify the subject matter experts for a software project or module within a project. Several techniques have been explored in the past to extract software metrics from a version control repository, and each is specific to the data being sought. This paper describes a network analysis technique that can be used to accurately identify the core developers for a specific software project, and measure how often the developers work together on the same area of code. The analysis was performed on 303 open source projects. Specific details were presented for 3 of these projects, and the general trends were identified based on the analysis of 303 projects. The accuracy of this analysis was confirmed based on credits and other information available on the project websites. Information related to communities of authors within a project was difficult to verify.

The information gathered from this analysis is useful for a new developer in order to identify subject matter experts to answer their questions, and for a project manager when assigning resources. The clustering coefficient of the author graph is a useful indicator for a project manager. If the clustering is too low, then there may be increased risk of key team members leaving the organization. The distribution of clustering coefficients of all projects can be used by a project manager as a basis of comparison.

This analysis should be expanded in the future to attempt to improve the accuracy of the results and to obtain more insight into the project structure. One area that can be explored is how the data is filtered. This study allowed a user to filter the data by

file name and to exclude the original addition of files to the repository. The ability to filter by a time period would be useful to limit the analysis for long-lived projects. Other filtering techniques could be developed to limit the analysis to only a certain set of authors, or files containing certain text.

Another area of future research is analyzing how the author graph changes over time. As new developers start work on the project, how do they get incorporated into the author network, and how to older developers transition away from a central role? This may offer insight into the team dynamics for a project and indicate how accepting they are of new developers.

References

1. Collins-Sussman, B., Fitzpatrick, B.W., Pilato, C.M.: Version control with subversion (2007). URL http://svnbook.red-bean.com/en/1.4/index.html
2. Howison, J., Crowston, K.: The perils and pitfalls of mining sourceforge. In: In Proceedings of the International Workshop on Mining Software Repositories (MSR 2004, pp. 7–11 (2004)
3. Huang, S.K., Liu, K.m.: Mining version histories to verify the learning process of legitimate peripheral participants. In: Proceedings of the 2005 international workshop on Mining software repositories, MSR '05, pp. 1–5. ACM, New York, NY, USA (2005)
4. The igraph website (2010). URL http://igraph.sourceforge.net/
5. Kagdi, H., Yusuf, S., Maletic, J.I.: Mining sequences of changed-files from version histories. In: Proceedings of the 2006 international workshop on Mining software repositories, MSR '06, pp. 47–53. ACM, New York, NY, USA (2006)
6. Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., Baldi, P.: Mining eclipse developer contributions via author-topic models. In: Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on, pp. 30 –30 (2007). DOI 10.1109/MSR.2007.20
7. Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M.: Applying social network analysis to the information in cvs repositories. In: Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on (2004). DOI 10.1109/ICSE.2004.1317529
8. Newman, M.E.J.: Coauthorship networks and patterns of scientific collaboration. Proceedings of the National Academy of Sciences of the United States of America **101**(Suppl 1), 5200–5205 (2004). DOI 10.1073/pnas.0307545100. URL http://www.pnas.org/content/101/suppl.1/5200.abstract
9. Newman, M.E.J.: Networks an Introduction. Oxford University Press, New York, NY (2010)
10. Ordonez, M., Haddad, H.: The state of metrics in software industry. In: Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on, pp. 453 –458 (2008). DOI 10.1109/ITNG.2008.106
11. Reichardt, J., Bornholdt, S.: Statistical mechanics of community detection. Phys. Rev. E **74**(1), 016,110 (2006). DOI 10.1103/PhysRevE.74.016110
12. Sommerville, I.: Software Engineering, 8. edn. Addison-Wesley, Harlow, England (2007)
13. Umarji, M., Shull, F.: Measuring developers: Aligning perspectives and other best practices. Software, IEEE **26**(6), 92 –94 (2009). DOI 10.1109/MS.2009.180
14. Voinea, L., Telea, A.: Mining software repositories with cvsgrab. In: Proceedings of the 2006 international workshop on Mining software repositories, MSR '06, pp. 167–168. ACM, New York, NY, USA (2006)
15. Voinea, L., Telea, A.: An open framework for cvs repository querying, analysis and visualization. In: Proceedings of the 2006 international workshop on Mining software repositories, MSR '06, pp. 33–39. ACM, New York, NY, USA (2006)