Mining and visualizing developer networks from version control systems

ARTICLE · JANUARY 2011

DOI: 10.1145/1984642.1984647

Mining and Visualizing Developer Networks from Version Control Systems

Andrejs Jermakovics
Free University of Bozen-Bolzano
Piazza Domenicani 3
Bolzano, Italy
Andrejs.Jermakovics@unibz.it

Alberto Sillitti
Free University of Bozen-Bolzano
Piazza Domenicani 3
Bolzano, Italy
Alberto.Sillitti@unibz.it

Giancarlo Succi
Free University of Bozen-Bolzano
Piazza Domenicani 3
Bolzano, Italy
Giancarlo.Succi@unibz.it

ABSTRACT
Social network analysis has many applications in software engineering and is often performed through the use of visualizations. Discovery of these networks, however, presents a challenge since the relationships are initially not known. We present an approach for mining and visualizing networks of software developers from version control systems. It computes similarities among developers based on common file changes, constructs the network of collaborating developers and applies filtering techniques to improve the modularity of the network. We validate the approach on two projects from industry and demonstrate its use in a case study of an open-source project. Results indicate that the approach performs well in revealing the structure of development teams and improving the modularity in visualizations of developer networks.

Categories and Subject Descriptors
D.2.9 [Software Engineering]: Management – programming teams. H.5 [Information Interfaces and Presentation]: Group and Organization Interfaces – Collaborative computing, Computer–supported cooperative work

General Terms
Management, Measurement, Experimentation, Human Factors

Keywords
Software processes, social networks, visualization

1. INTRODUCTION
Social network analysis has many applications in software engineering and has been applied to developer networks for exploring collaboration [31], predicting faults [25], studying code transfer [22] and many other activities. Moreover, since software artifact structure is strongly related to the organization's structure (*Conway's Law* [6]) it becomes important to understand the developer networks involved. The analysis of these networks is often leveraged using visualizations and their appearance plays a significant role in how people interpret the networks [17, 20]. It is, therefore, important that the collaboration network visualizations are easy to interpret and represent the actual network as closely as possible.

A common problem is that the actual social networks are not known and need to be discovered. Many existing approaches rely on communication archives to discover the networks [2, 7, 27]; however these are not always available. Another convenient source of developer networks is the Version Control Systems (VCS) [9, 16, 29]. The underlying idea is that frequent access and modification on the same code implies communication and sharing. The advantages of using VCS is that they are commonly available for all software development activities, can be mined automatically without human involvement and directly reflect collaboration on code.

Once a developer network is constructed it can be analyzed to improve understanding of the software development process and the organizational structure. The recovered organizational structure then can be used in informing new management or observing the integration of new team members. Additional applications include tracking code sharing, finding substitute developers with related code knowledge and assembling new teams with prior collaboration experience.

In this paper we present a novel approach for discovering and visualizing collaboration networks of software developers from VCS. The main contributions of the paper are that developer links are mined from changes at file level and that link strength is computed using Cosine similarity. Additionally, we compare the performance of multiple filtering techniques for increasing the modularity of networks. We describe our approach for constructing the networks, validate it on two industrial company data and perform a case study on an open-source project.

2. RELATED WORK
There has been a lot of recent research on the usage of social network analysis in software engineering. Since our work is focused on the discovery and visualization of developer networks, in the related work, we look at other approaches involving these aspects.

Similarly to our work Lopez-Fernandez et. al. [9] collect developer links from CVS repositories and propose social network analysis for characterizing open-source projects. The difference is that they focus on analyzing characteristics of developer networks and not on analysis using visualizations. In addition, the data collection is based on developer commits to CVS modules, thus more coarse grained, and the weight of developer-developer links is not reduced for modules with many developers.

ICSE'11, May 21-28, 2011, Waikiki, Honolulu, HI, USA.

Also Huang and Liu [16] use CVS repositories to mine developer networks and discover core and peripheral developers in open-source projects using distance centrality measure. A link between developers is defined if they contributed to the same module and, unlike our approach, the links are unweighted. Because we mine relationships at file level, the resulting networks are dense and, in these cases, we find link weighting to be a necessity. The authors acknowledge that grouping developers at directory level is possibly not detailed enough because for some projects the structure was not evident.

In Ariadne [30] a call-graph is extracted from Java programs and is annotated with author information to build sociograms of developers. Thus, the dependencies among modules are translated to dependencies of their corresponding authors and the strength of the developer dependency is based on code dependencies. Our work, on the contrary, defines relationships based on common code changes and is not aimed at dependency analysis.

Tesseract [27] is a system for exploring socio-technical relationships and extracts developer links from version control systems. Similarly to the proposed approach it displays a developer network with a link defined if developers edited the same artifact and offers a filtering threshold. While force-directed layout is used here as well, the weight of links is not based on the number of changes but on the amount of communication between developers (represented by edge width). Thus the approach requires communication archives and the focus is on communication whereas the focus of this work is collaboration on code.

In [2], Bird et al. analyze social networks of open-source project developers to determine patterns of organization. They extract social networks from email communication and then find communities by computing a graph partitioning that maximizes a modularity measure [23]. In our experience the developer networks mined from VCS are very dense (with low modularity) therefore this method was not applicable.

Ohira et al. [24] use collaborative filtering and social network visualizations to show developers of open-source projects. The difference between our work and theirs is that they use developer participation in projects to identify links, while we use file change history. Thus, our link identification is more fine grained and can be applied to participation in a single project as well as multiple projects. Another difference is that they do not visualize low similarity relations to avoid the complexity, however, we chose to preserve them if these links are important to the structure. They have used hyperbolic graph visualizations for displaying the networks which we have not experimented with in this work.

Augur [29] is a visualization tool that, together with other repository visualizations, shows a developer network in the form of a graph. The similarity between developers is based on changes to common CVS modules and is represented by the thickness of edges. In contrast, we compute similarities at file level independent of module and have chosen edge lengths for representing similarity in order to let the graph layout outline groups and team structure.

Overall, multiple developers network visualizations exist but have different aims, data collection approaches, link computation and visualization methods. Because of these variations and due to a lack of common benchmarks we were not able perform an objective comparison of all the approaches. Instead we argue that this work contributes to the existing work with a novel and more general approach enabling a wider use of developer network visualizations. It builds networks from file change histories; therefore, it is easily applicable in the majority of development environments. However, this choice of data source required us to focus on similarity measures and filtering subtleties to improve the analysis.

3. PROPOSED APPROACH

Our proposed approach uses VCS (Subversion, CVS, Team System) to mine commits to source code files that developers make. It then computes similarities between committers and visualizes them in a network using similarities as link strengths. In cases when the network is too dense it offers multiple filtering techniques to reduce the number of links.

3.1 Similarity Measure

A crucial part of our approach is the similarity measure because it is used as the basis for visualization and filtering. For our purposes, we adopted an established similarity measure which is also used in Collaborative Filtering techniques [26, 28] of Recommender Systems. A number of user similarity measures are derived from the ratings that users assign to items. In our case, we use source files as the items and the number of changes as the rating which is similar to an approach for recommending software components [20].

Each developer is treated as a vector holding weights for all files. The weight of a file f for a developer d is defined using the number of commits to f by d:

$$weight_{f,d} = commits_{f,d} \cdot log(N/n_f)$$

in which N is the total number of developers and n_f is the number developers that have changed f. The purpose of $log(N/n_f)$ is to lower weights for files that many developers have modified hence reducing their impact on similarities. Cosine similarity between two developers is obtained by calculating the cosine of the angle between their corresponding vectors a and b:

$$similarity(a,b) = \cos(\mathrm{a},\mathrm{b}) = \frac{a \cdot b}{|a||b|} = \frac{\sum_k a_k\, b_k}{\sqrt{\sum_k a_k^2 \sum_k b_k^2}}$$

The similarity accounts for files that were modified only a few times and also for files that are modified by many people. Thus it is greater if the two developers modified the same files a large number of times and 0 if they did not share any files. To our knowledge, such similarity measure has not yet been used in constructing and visualizing developer networks from common file changes.

It should be noted that other similarities can be computed on the developer vectors and we compare it to Jaccard similarity and Pearson's correlation in a later section. As the experiments show good results have been observed with Cosine similarity therefore we use it as the main similarity measure for constructing developer networks.

3.2 Visualization

A common choice for social network visualizations [15, 27] is to use multidimensional scaling [3, 11] or force-directed graph layout algorithms [12]. Such algorithms model graph vertices as having physical forces of attraction and repulsion. They iteratively compute vertex positions until the difference between desired and

actual distances is minimized. Their advantage is that groups with high connectivity are placed together and similar vertices are placed closer than dissimilar ones.

We apply Fruchterman-Reingold [12] force-directed graph layout to the constructed network by setting the link lengths to developer dissimilarity in order to place similar developers together and dissimilar ones apart. The size of each node in the network is proportional to the number of commits the developer made and no link is created between developers with similarity 0. In cases when the visualization appears complex due to a large number of links, we apply filtering to remove low similarity links.

For visual appeal, links are visualized with transparency (using alpha blending). The transparency of a link is proportional to the similarity – high similarity links are solid while low similarity links are transparent. Thus strong links can be spotted immediately and the viewer can see which edges will disappear first during filtering.

3.3 Filtering

In our experience, developer networks constructed from common file changes contain a large number of links. Most of the links, however, have a very low weight and their presence can be explained by small changes to common files. Such complex network with many links is harder to interpret and not all links might be of immediate interest to the viewer. Moreover, force-directed graph layout becomes more constrained in preserving distances and graph clustering algorithms perform worse on dense graphs [4]. We investigate several filtering techniques to reduce the number of links:

3.3.1 Global Low Similarity Filtering

The most used filtering approach is to remove links with lowest weights in the whole network. It can make the network clearer, however, our experience also shows that this can leave some nodes disconnected from the rest of the network. Also, groups that are connected by light links will lose their links first and the group may become disconnected while other groups are still strongly connected.

3.3.2 MST Constrained Filtering

We propose a Minimum Spanning Tree (MST) constrained filtering technique which removes low similarity links as before, but does not remove links that are part of the Minimum Spanning Tree (MST) [19] of the current network. The advantage is that the filtered network remains connected and the viewer can see at least the strongest link of every node. When clusters emerge during filtering then this method helps to see which people connect the clusters.

3.3.3 Local Low Similarity Filtering

We propose another filtering technique in which the lowest similarity links of each developer are removed first. As the filter is increased, each node initially loses their least important links and important links are kept (larger weight). While with MST constrained filtering a single strongest link remains, this method allows keeping several strong links. Thus the network can contain weakly and strongly connected groups and at the same time be comprehensible.

3.3.4 High Betweenness Centrality Filtering

Betweenness Centrality (BC) [1, 10] of a vertex or edge is defined as the number of shortest paths between pairs of vertices that run through it. For a graph $G = (V, E)$ and an edge $e \in E$ the BC is:

$$BC(e) = \sum_{i \neq j \in V} \frac{\sigma_{ij}(e)}{\sigma_{ij}}$$

where σ_{ij} is the number of shortest paths that run from vertex i to j and $\sigma_{ij}(e)$ is the number of shortest paths going through e.

Edge betweenness has been used for graph clustering [23] and identifying community structure in various networks (including collaboration networks) by iteratively removing the edge with the highest BC [13] and then recomputing the BC measures again. The removal of edges with high BC allows clusters to separate, however this approach works better on sparse graphs than on dense ones.

3.3.5 SimRank Based Filtering

SimRank [14] is an algorithm for computing similarities of nodes in graphs and is iteratively computed based on the similarities of neighboring nodes. In short, two nodes are similar if their neighboring nodes are similar. We use the described similarity as weight of links and initialize the algorithm with a row-normalized adjacency matrix of the network. With the suggested number of iterations and parameters we obtain similarities between all pairs of nodes. The filter then removes the links between least similar nodes (developers) thus keeping the links of most similar ones.

Each filtering technique requires a threshold parameter which is specified in our implementation using a slider control. Thus the visualization can be experimented with and interactively adjusted.

3.4 Implementation

The described visualization is implemented in our software visualization and analysis tool, Lagrein [18], which we used for experimentation. The tool provides an interactive environment for importing and analyzing developer networks. Besides the choice of visualization method and filtering techniques, it provides several interaction features for more detailed analysis.

3.4.1 Committed File View

By selecting a link or a group of developers we can see the files that this group worked on. The files are presented in sorted order of their combined weights thus showing at first the most significant files for that group. This view proved very helpful during our experiments by providing additional insight.

3.4.2 File Subset Selection

The user can select a subset of all files to be included in the analysis. This allows focusing ones attention on collaboration in a particular part of the project or a subsystem. Also this allows excluding files that we do not want to be considered in link computation (e.g. non source code artifacts).

Overall, the whole network can be interactively explored for additional information after the initial visualization. After selecting a node the tool shows information on that developer (number of commits, first-last commit date, node degree) and by selecting a link the tool shows the similarity.

4. VALIDATION

We validate our approach on data from two industry companies using their VCS and information on their team structure which we gathered from team leaders. We compare the structure that the method shows against the actual known structure and the goal of the validation is to ensure that they match. Once this is established we would be able to use it on other projects for which the structure is not known.

We perform the validation in several steps: first we validate the similarity measure and then compare the different filtering techniques. Due to nondisclosure agreements we omit the names of the companies and developers.

We use the *modularity measure* [23] for comparing the performance of similarity measures. Intuitively, given a graph and known clusters modularity has a higher value if the links within the clusters are dense and links among clusters are sparse. It can take negative values in case of unmodular networks and for perfectly modular can reach *1-1/n* where *n* is the number of clusters [8].

The measure is computed by considering a partitioning of the network into k clusters and constructing a $k \times k$ symmetric matrix, e. The value of each element e_{ij} is the fraction of all links that connect cluster i to cluster j. Using the row sums $a_i = \sum_j e_{ij}$ the modularity measure is defined as:

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

In our experiments we use known teams as clusters and compute the modularity knowing that there should be more links within teams than across. Thus modularity allows us to compare how well the teams are separated from each other during filtering. Higher modularity indicates better performance and makes it easier to distinguish the teams in the visualized network.

Modularity only partially measures how well the created network represents the real life network; however we consider the ability to identify teams an important aspect of evaluation. Therefore at first we only validate in regard to this aspect and acknowledge that further validation is needed afterwards.

Company A. Our first industry partner is a Northern European company working in the financial sector. The company has two development teams of 20 people in total and is working on a large distributed system with a size of 500 KLOC. One of the teams is working on server-side components in C++ and the other on client-side components written in Java. While the teams work mostly on separate code, they also collaborate on integration of server and client side components. Thus we expect to see both teams as connected components in the visualization and some links between the teams. The data was extracted from their Subversion (SVN) repository for a period of last 2 years.

Company B. Our second industry partner is an IT department of a large Italian company. Their main responsibilities are developing and maintaining custom software that is needed by other departments of the company written in C#, C++ and Visual Basic with a total size approaching 700 KLOC. There is a main development team, second smaller team and one remote team in another country. Their development style is agile with collective code ownership therefore we expect to see a lot of code sharing; however more code should be shared inside each team. Former collaboration with the company is described in another paper [5].

The developer network was constructed from commit logs of Microsoft Team System VCS over a period of over 2 years and contains 52 committers.

4.1 Similarity Measure Comparison

The goal of similarity measure comparison was to establish which similarity can reveal the known developer teams the most. We compare the described Cosine similarity and similarities based on common files count. We also add two well known similarity measures for comparison:

- Common File Count
- Common File Commit Count
- Cosine Similarity
- Jaccard Similarity
- Pearson's Correlation

The *Common File Commit Count* has been used in other approaches and is computed by summing up the number of commits that two developers made to shared files. This comparison allows us to see whether the usage of our proposed similarity measures produces improvement.

Since in each project we know the actual teams, we can compute modularity of the network using the teams as clusters. Our and other approaches use filtering to remove low weight links; therefore we compute the modularity after each filtered link. The filtering method used is *Global Low Similarity* filtering because it is the simplest one and appears in other studies. We start with the lowest weight link first and iteratively remove the next low weight link until no links remain. This procedure simulates what modularity would be achieved by interactively using the filter in a tool.

4.1.1 Company A

From the similarity distribution we see that there is a large number of links however most of the links have very low weight due to low similarity value (Figure 1).

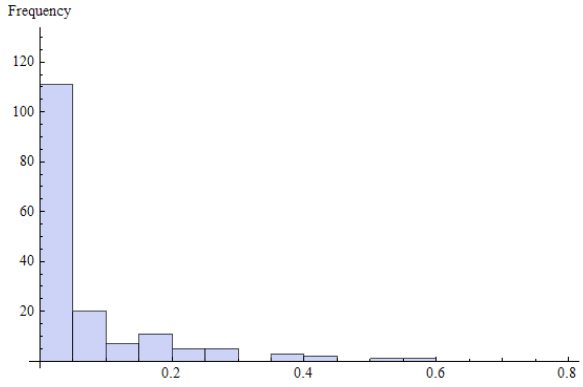

Figure 1. Link weight distribution using Cosine similarity (Company A). Most links are very light (<0.05)

Since the developer networks are initially dense, the modularity is low. As the low similarity links are removed, the modularity gradually improves for both measures (Figure 2). At the very end modularity significantly drops however this happens when almost all edges have been removed.

We further look at the initial and resulting visualizations obtained with each technique. Each team is colored in its own color and we also indicate each known team with a surrounding ellipse.

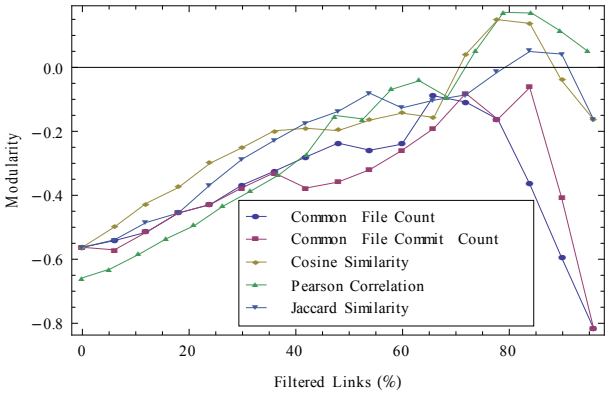

Figure 2. Filtering effect on modularity (Company A). Cosine similarity and Pearson correlation yield greater modularity

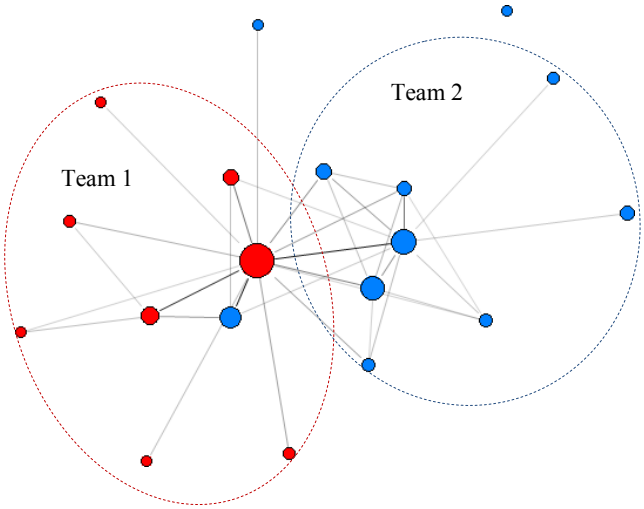

Figure 3. Initial network with commit sum similarity and edge filtering (Company A). Teams appear very close together

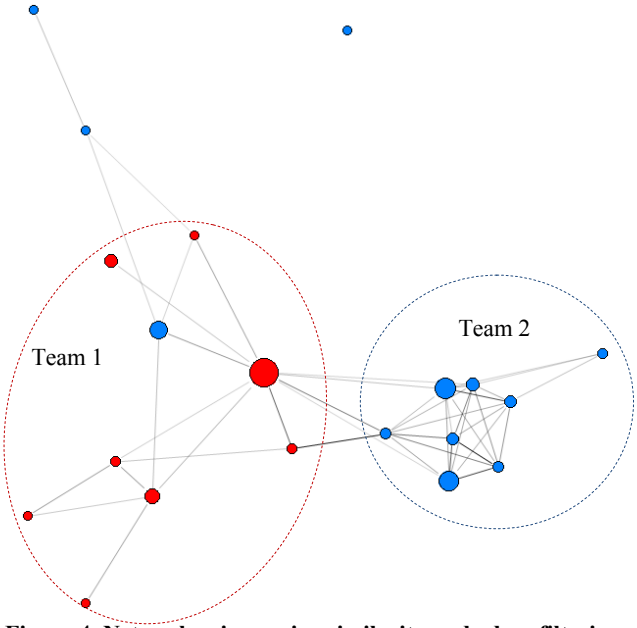

Figure 4. Network using cosine similarity and edges filtering (Company A). Teams become distinguishable.

Visualizations obtained with commit sum similarity initially do not to exhibit a clear structure (Figure 3). We see the teams on different sides; however they are too close together to be distinguishable.

With cosine similarity and filtering we are able to see the teams separate and with only the strongest links among them remaining (Figure 4). The developers placed in-between the teams were confirmed to be the ones working on client-server integration code. Moreover, the visualization shows that one team is more densely connected than the other suggesting that it shares code more. This information can be helpful in deciding how easy developers can substitute each other.

4.1.2 Company B

Also in the case of Company B we see a very large number of low similarity links.

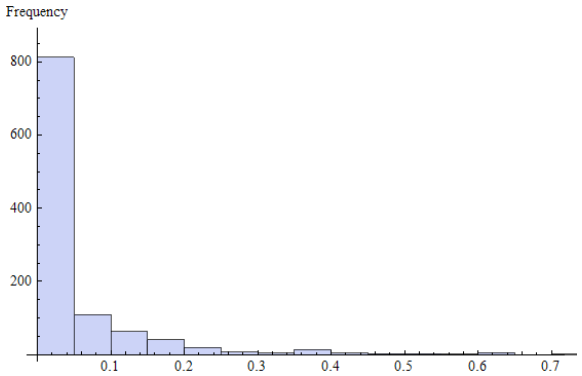

Figure 5. Link weight distribution using Cosine similarity (Company B). Most links are very light (<0.05).

As before we have a large number of low similarity links (Figure 5) and by gradually removing them we improve the modularity of the network (Figure 6).

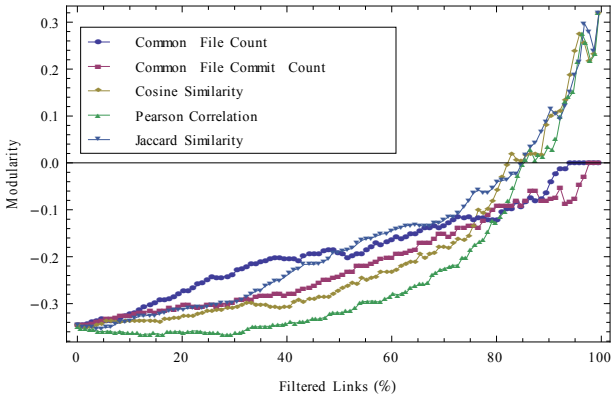

Figure 6. Filtering effect on modularity (Company B). Cosine and Jaccard similarity yield higher modularity.

Next we look again at the initial visualization and the resulting one. We experience a similar result as in the first experiment.

After applying Cosine similarity and filtering we are able to spot the teams in the network (Figure 8). The result is not perfect since the remote team is completely detached from the rest and the other two teams are slightly mixed, however we consider it a significant improvement over the initial visualization.

4.1.3 *Results*

For both companies the Cosine and Jaccard similarity achieve higher modularity rates with fewer removed edges. Pearson correlation performs well when most initial edges are removed but worse when removing few edges. They also achieve higher modularity in total. We conclude from these observations that the described approach can produce networks with higher modularity thus revealing clusters of developers.

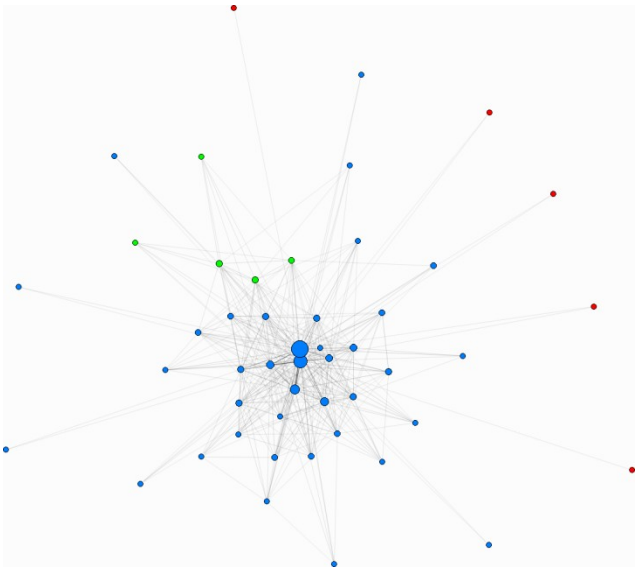

Figure 7. Initial network with commit sum similarity and edge filtering (Company B). No clear team structure is evident.

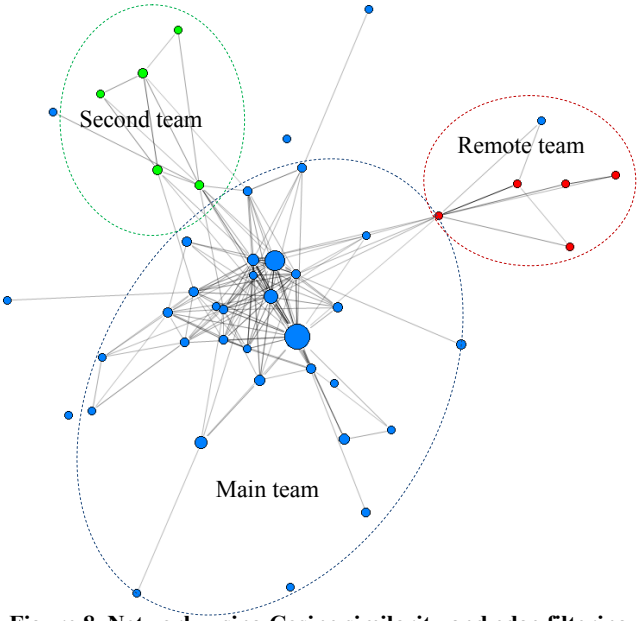

Figure 8. Network using Cosine similarity and edge filtering (Company B). Teams become distinguishable.

4.2 Filtering Method Comparison

Using the *Cosine similarity* measure we further compared the described filtering methods to determine which eventually yields higher modularity when applied.

4.2.1 *Company A*

For Company A the modularity is improved by all measures (Figure 9). The SimRank filtering method achieves almost maximum modularity in total, however it happens at a point where very few edges remain and half of all nodes are disconnected from the network.

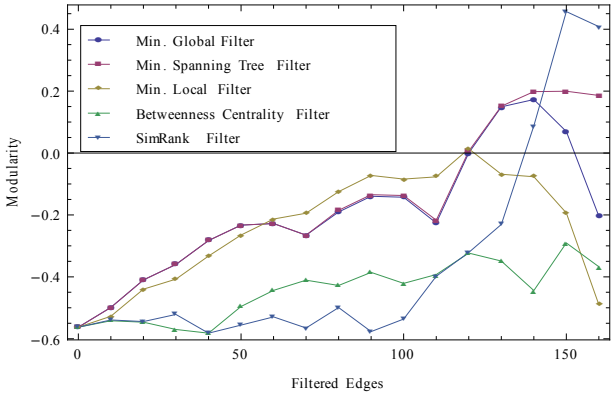

Figure 9. Filtering comparison (Company A). Higher performance for Global and MST filtering.

4.2.2 *Company B*

For Company B we see a similar scenario where the modularity is increasing for 3 filtering methods. This time we see a high spike from the BC filter, however only when very few edges remain.

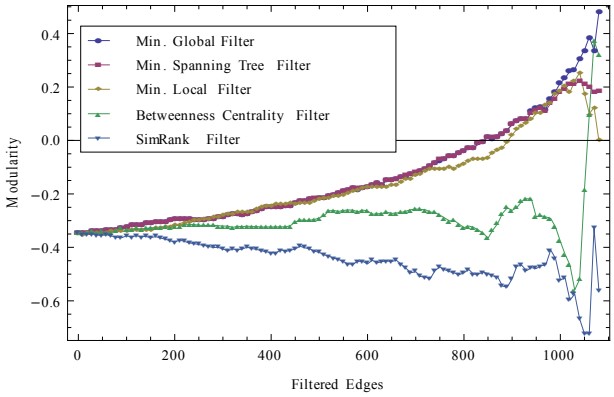

Figure 10. Filtering comparison (Company B). Higher performance for Global, Local and MST filtering.

4.2.3 *Results*

Both results show similarly high performance from Global, Local and MST filters and lower performance from BC and SimRank filters which can be attributed to the high density of the networks. It should be noted that we only use the default values of SimRank parameters. While the Global filtering is the simplest of all techniques, its downside is that it eventually disconnects the network (Figure 8). We therefore select *MST filter* as the preferred filtering method for our visualizations.

5. CASE STUDY: ECLIPSE DTP PROJECT

Having experimentally selected Cosine similarity and MST filtering as effective methods for discovering team structure we proceeded to apply the technique to the Eclipse Data Tools Platform (DTP) project[1]. The goal of the study is to demonstrate the use of the approach and to show that it can reveal aspects of the organizational structure that are not known beforehand.

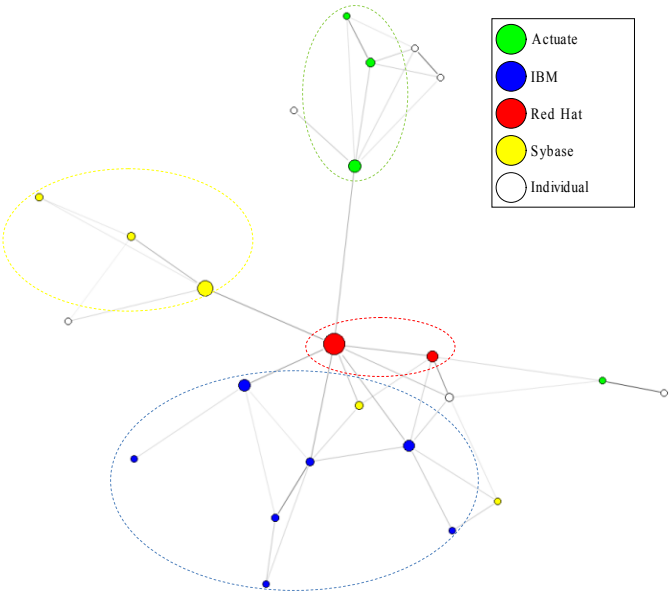

Figure 11. Eclipse DTP project committer network. There is higher collaboration of contributors within each company.

DTP is a set of tools for database handling and is currently part of Eclipse IDE distribution. The project is large (1.4 MLOC) and is composed of several subprojects: Connectivity, Enablement, Incubator, Model Base and SQL Development Tools. Using its CVS repository we constructed the developer network of 25 people for the period 2005-2010.

The Eclipse project provides information on its committers using its Commits Explorer[2]. This application allowed us to learn that contributions to the project have been made by many individual committers and multiple companies including Actuate, IBM, Red Hat and Sybase.

Figure 11 shows the resulting network obtained with MST filtering and having each company colored in different color. The visualization of the network allows us to gain quick insight into the organizational structure of the project. Namely, contributors do not contribute equally to all parts of the project. They collaborate closely with other contributors from the same company and to a much lesser extent with contributors from other companies. This suggests that the collaboration is performed on specific subprojects.

6. LIMITATIONS

While we have observed promising results, we have also discovered several limitations of the approach.

First of all, our approach currently considers new and old commits having equal importance. The result is that long term past collaboration has higher impact on the similarity than short but recent collaboration. We experienced this scenario with Company A when a developer was moved from one team to the other but was still visualized in the middle of his old team (Figure 4).

Secondly, filtering techniques require manual adjusting of the threshold. Although this feature gives control over the number of visualized links; it can also be considered a limitation due to human involvement. This limitation could be overcome with an automatic threshold detection algorithm.

Thirdly, the link computation is based on an assumption that frequent collaboration on code implies communication. In some cases, however, there was a lot of communication which was not reflected by code changes. Company A had a developer working on plugins that interact with the rest of the system through existing interfaces. Since these interfaces did not require frequent modification, the link between the plugin developer and the rest of his team was very weak.

Lastly, we have to mention that file renaming and moving in the repository is not currently handled. A renamed file is considered as a different file and thus both files are used in similarity computation.

7. CONCLUSIONS AND FUTURE WORK

Social network analysis is often performed by relying on the visual representation of the network and one of the most convenient sources of software developer networks is VCS. We have presented an approach for constructing and visualizing developer networks from VCS based on common file changes. It first applies weighting to all files and then computes developer similarity using cosine measure. Since networks from VCS are typically very dense; we propose filtering to improve the modularity. We validate the similarity and several filtering methods by conducting experiments on industry data and demonstrate the use of the approach in a case-study. The results indicate that the approach is able to improve the modularity in developer networks and can reveal the underlying team structure.

Our future work will focus on evaluating our approach using other measures than modularity and on addressing the limitations of the approach. Namely, we are interested in adjusting the similarity measure to discard commits that are too far apart.

8. REFERENCES

[1] Anthonisse, J. M. 1971. The rush in a directed graph. Technical Report BN9/71, Stichting Mahtematisch Centrum, Amsterdam.

[2] Bird, C., Pattison, D. and D'Souza, R. 2008. Chapels in the Bazaar? Latent Social Structure in OSS, in 16th ACM SigSoft International Symposium on the Foundations of Software Engineering, (Atlanta, GA)

[3] Borg, I. and Groenen, P. 1997. Modern Multidimensional Scaling: Theory and Applications, Springer-Verlag.

[4] Brandes, U., Gaertler, M. and Wagner, D. 2003. Experiments on graph clustering algorithms. In Proc. 11th Europ. Symp. Algorithms (ESA '03), LNCS 2832, pages 568--579. Springer-Verlag.

[5] Coman, I. D., Sillitti, A., and Succi, G. 2009. A case-study on using an Automated In-process Software Engineering

[1] http://www.eclipse.org/datatools/

[2] http://dash.eclipse.org/

Measurement and Analysis system in an industrial environment. In Proc. of the 2009 IEEE 31st International Conference on Software Engineering (May 16-24, 2009), IEEE Computer Society, Washington, DC, 89-99

[6] Conway, M. E., "How Do Committees invent?", Datamation, 14(4):28-31, April 1968

[7] Crowston, K. and Howison, J. 2005. The Social Structure of Free and Open Source Software. First Monday, 10(2), 2005.

[8] Danon, L.,Duch, J., Diaz-Guilera, A. and Arenas, A., Comparing community structure identification, J. Stat. Mech. P09008 (2005).

[9] L. Lopez-Fernandez, G. Robles, and J. M. Gonzalez-Barahona. Applying social network analysis to the information in CVS repositories. In Proc. of 1st Intl. Workshop on Mining Software Repositories (MSR2004), pages 101–105, 2004.

[10] Freeman, L. C. 1977. A Set of Measures of Centrality Based on Betweenness, Sociometry 40, pp. 35-41.

[11] Freeman, L. C. 2000. Visualizing Social Networks, Journal of Social Structure

[12] Fruchterman, T. M. G. and Reingold, E. 1991. Graph Drawing by Force-Directed Placement, Software-Practice and Experience 21, pp. 1129–1164

[13] Girvan M. and Newman M. E. J. 2002. Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821-7826

[14] Jeh G. and Widom J. 2002. Simrank: a measure of structural-context similarity. In KDD ’02: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 538–543.

[15] Heer, J. and Boyd, D. 2005. Vizster: Visualizing Online Social Networks. In Proc. of the 2005 IEEE Symposium on Information Visualization, page 5. IEEE Computer Society, (Washington, DC, USA), INFOVIS'05

[16] Huang, S.-K., Liu, K.-M. 2005. Mining version histories to verify the learning process of legitimate peripheral participants. Proceedings 2nd International Workshop on Mining Software Repositories (MSR’05). ACM Press: New York NY; 84–78.

[17] Huang, W., Hong, S., and Eades, P. 2006. How people read sociograms: a questionnaire study. In Proc. of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60. Australian Computer Society, 199-206.

[18] Jermakovics, A., Scotto, M., Sillitti, A., Succi, G. “Lagrein: Visualizing User Requirements and Development Effort”, 15th International Conference on Program Comprehension (ICPC 2007), Banff, Canada

[19] Kruskal, J. B. 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In: Proceedings of the American Mathematical Society, Vol 7, No. 1, pp. 48–50

[20] Mccarey, F., Cinnéide, M. Ó., and Kushmerick, N. 2005. Rascal: A Recommender Agent for Agile Reuse. Artif. Intell. Rev. 24, 3-4 (Nov. 2005), 253-276.

[21] McGrath, C. and Blythe, J. and Krackhardt, D. 1997. The effect of spatial arrangement on judgments and errors in interpreting graphs. Social Networks, 19, 223-242

[22] Mockus, A. 2009. Succession: Measuring transfer of code and developer productivity. In Proc. of the 2009 IEEE 31st International Conference on Software Engineering (May 16 - 24, 2009). IEEE Computer Society, Washington, DC, 67-77.

[23] Newman M. E. J. and Girvan M. 2004. Finding and evaluating community structure in networks. Physical Review, Vol. E 69 , Nr. 026113.

[24] Ohira, M., Ohsugi, N., Ohoka, T., and Matsumoto, K. 2005. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. SIGSOFT Softw. Eng. Notes 30, 4 (Jul. 2005), 1-5

[25] Pinzger, M., Nagappan, N., and Murphy, B. 2008. Can developer-module networks predict failures?. In Proceedings of the 16th ACM SIGSOFT international Symposium on Foundations of Software Engineering (Atlanta, Georgia, November 09-14, 2008). SIGSOFT '08/FSE-16. ACM, New York, NY, 2-12

[26] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of CSCW '94, Chapel Hill, NC.

[27] Sarma, A., Maccherone, L., Wagstrom, P., and Herbsleb, J. 2009. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In Proceedings of the 2009 IEEE 31st international Conference on Software Engineering (May 16 - 24, 2009). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 23-33

[28] Shardanand, U. and Maes, P. 1995. Social information filtering: algorithms for automating “word of mouth”. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM Press/Addison-Wesley Publishing Co., New York, NY, 210-217

[29] de Souza, C., Froehlich, J., and Dourish, P. 2005. Seeking the source: software source code as a social and technical artifact. In Proceedings of the 2005 international ACM SIGGROUP Conference on Supporting Group Work. GROUP '05. ACM, New York, NY, 197-206.

[30] de Souza, C. R., Quirk, S., Trainer, E., and Redmiles, D. F. 2007. Supporting collaborative software development through the visualization of socio-technical dependencies. In Proceedings of the 2007 international ACM Conference on Supporting Group Work. GROUP '07. ACM, New York, NY, 147-156

[31] Wolf, T., Schröter, A., Damian, D., Panjer, L. D., and Nguyen, T. H. 2009. Mining Task-Based Social Networks to Explore Collaboration in Software Teams. IEEE Softw. 26, 1 (Jan. 2009), 58-66.