Multiple Social Networks Analysis of FLOSS Projects using Sargas

Samuel F. de Sousa Júnior
Universidade Federal do Pará, Brazil
sfelixjr@gmail.com

Marco A. Balieiro
Universidade Federal do Pará, Brazil
ma.balieiro@gmail.com

Jean M. dos R. Costa
Universidade Federal do Pará, Brazil
jeanmrc@gmail.com

Cleidson R. B. de Souza
Universidade Federal do Pará, Brazil
cdesouza@ufpa.br

Abstract

Due to their characteristics and claimed advantages, several researchers have been investigating free and open-source projects. Different aspects are being studied: for instance, what motivates developers to join FLOSS projects, the tools, processes and practices used in FLOSS projects, the evolution of FLOSS communities among other things. Researchers have studied collaboration and coordination of open source software developers using an approach known as social network analysis and have gained important insights about these projects. Most researchers, however, have not focused on the integrated study of these networks and, accordingly, in their interrelationships. This paper describes an approach and tool to combine multiple social networks to study the evolution of open-source projects. Our tool, named Sargas,allows comparison and visualization of different social networks at the same time. Initial results of our analysis can be used to extend the "onion-model" of open source participation.

1. Introduction

Free/libre and open source software development (FLOSS) is an approach in which loosely-knit collections of volunteers, collaborating over the public Internet, create software systems whose source code is available to all. There are several claimed advantages of this approach, including faster development cycles, and more secure and robust software products. Due to these advantages and other characteristics, several researchers have been investigating open-source projects and communities. They want to find out what motivates developers to join FLOSS projects, the tools, processes and practices used in FLOSS projects [5,6], the evolution of FLOSS communities [7,8], among other aspects of these successful projects.

In particular, researchers have studied collaboration and coordination of open source software developers using an approach known as social network analysis [13]. Social network analysis allows one to study the relationship among developers and/or projects to understand their structural properties. For instance, Lopez-Fernandez et al. [2] have looked at what they call "committer networks": social networks where each vertex corresponds to a particular committer (a software developer) and two committers are connected when they have contributed to at least one common module. This is one type of social network that has been used by researchers. There are several others that model a different relationship between software developers, i.e., a different aspect of a FLOSS project. This has been very fruitful, since research using social network analysis methods has provided important insights about FLOSS projects.

However, researchers have not focused on the integrated study of different social networks, i.e., previous research applying SNA to FLOSS projects has looked at interrelated aspects in isolation. The work described in this paper aims to alleviate this limitation by combining multiple social networks to study the evolution of open-source projects, i.e., to find out how these different social networks influence or are related to each other. This work extends our ongoing work on software tools to study open-source communities [18]. In this paper, we describe Sargas, a multiple social networks visualization tool that allows comparison and visualization of different social networks metrics at the same time through a StarPlot-based [10] visualization.

The rest of this paper is organized as follows. Section 2 presents the motivation for our work. Next, the Sargas visualization tool is presented. In Section 4 we present our methodology which includes a brief social network analysis introduction, the description of the project used, and the other tools (TransFlow and OSSNetwork) used to generate the data needed by Sargas. In Section 5, we present the results and discussion of the PMD project used as a case study in this paper. Finally, conclusions and future work are presented.

2. Motivation

Researchers from different areas ranging from software development to economics and social science have been studying FLOSS projects. As expected, different research approaches have been used including interviews, surveys, regression models, case studies, and so on. In this paper, we are mostly concerned with the usage ofsocial network analysis methods [13] by researchers interested inthe coordination of FLOSS projects. Social network analysis allows one to study the relationship among developers and/or projects to understand their structural properties. An example of this type of research is the work by Crowston and Howison [17] who classify the network of developers involved the bug-fixing tasks: it contains core developers, co-developers who provide bug fixes and active users who report bug fixes. In addition, Lopez-Fernandez et al. [2], in contrast, have looked at committer networks: networks where each vertex corresponds to a particular committer and two committers are connected when they have contributed to at least one common module. Gloor [3] has looked at networks of developers created from emails exchanged among open source developers in order to identify how innovation is achieved in open source communities. Finally, de Souza et al. [4] have looked at social networks extracted from the source code to identify transitions from the periphery to the core of a project, and vice-versa. These are only a few examples of SNA approaches for the study of open source projects. Others include [1] and [9].

Each one of these researchers has provided important insights into the phenomenon of collaborative software production in open/free software projects. In fact, each one of these networks represents a different relation between software developers. However, researchers have not focused on the combined study of these networks. That is, by looking at individual aspects of a FLOSS projects, it is only possible to partially understand these projects. For instance, Crowston's and Howison's [17] work broadly divides contributors into "developers" (who write part of the code) and "users" (who might report bugs), while in practice, "developers" are also "users", that is, contributors who write code also report bugs in the software. By looking at different social networks (the bug fixing network, the source-code network, the network extracted from the mailing lists and so on) at the same time, it is possible to extend the roles performed by FLOSS contributors to more clearly reflect the everyday practice of these contributors. In short, the contribution of this paper is an approach and tool to combine different social networks to understand FLOSS projects.

In the following section, we describe Sargas, the tool that instantiates our approach and analyzes multiple social networks at the same time.

3. Sargas

Sargas is a multiple social networks visualization tool developed in Java. The visualization method used in Sargas is based on the StarPlot [10] visualization. In this approach, a "star" is created for each user so that each face of the "star" presents information about one specific social network. Currently, we present centrality measures [13] in each face of a star, so that it is possible to represent multiple measures (one for each social network) in a single representation. For instance, it is possible to find out whether a high centrality developer found in a source code network also has a high centrality in the communication network through the visual inspection of the two faces of this developer's star. Furthermore, when different developers (and their associated stars) are aligned, it is possible to easily compare the contribution of several developers in different social networks at the same time.

Figure 1 below presents an example of a Sargas-generated visualization. In this visualization, we draw a white circle around each star to indicate where the mean-value is located, therefore, it is possible to observe that actor user A has a centrality measure below the average in the social network represented by the yellow face (pointing north). In contrast, user B has a centrality value above the average for the same social network. That is, user B is more central than user A in the social network represented by the yellow face(north) of the star.

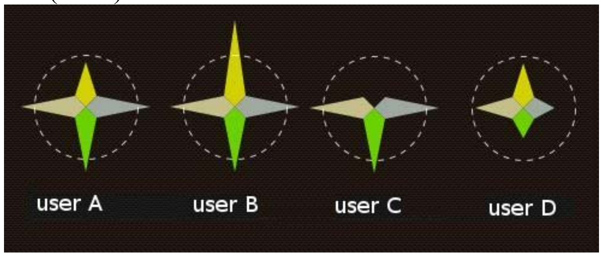

Figure 1 – An example of Sargas visualization.

Sargas works as follows. Its inputs are social networks that can be extracted from dl or csv files. For each social network, a centrality measure (currently, degree, betweenness or information) can be selected. A network generated by discussion about bugs or chat conversations can be analyzed, for instance, using betweenness centrality, allowing one to find the potential developerswho are located in "between" other developers, and, therefore who have some control of the communication between two nonadjacent actors [13]. On the other hand, in a social network extracted from the source-code, it might be necessary to calculate the degree centrality of each software developer to understand how influential it is in the structure of the code. In short, with Sargas it is possible to use a different centrality measure for each social network being analyzed.

One needs to be careful when comparing different social networks: a social network generated from source code can be quite different from a bug network or mailing list network. For instance, if an actor has a degree value of 120 in the chat conversations network, but it has a value of centrality measure of 20 in a source-code network, it would not be appropriate to compare these values because they are based on different measures of different social networks. To overcome this problem, Sargas uses a statistical approach, the z-score, to compare different distributions. A z-score is computed for each centrality measure of each software developer in each social network. Z-scores are calculated based on the average and standard deviation of each (social network) distribution, so that they allow a particular centrality value to be referenced by the number of standard deviations it is above or below the mean of that social network. In other words, because we use z-scores, it is possible to compare different values of different metrics because these values are transformed into new values that describe how far the original values are from the average value calculated for each network.

In Sargas visualization, the number of faces of the star can vary. Accordingly, the degree between two faces is calculated dynamically. For instance, for three faces (i.e., three social networks), the degree between these faces will be 120º, for four faces, it will be 90°, and so on.

4. Methodology

4.1 Social Network Analysis

A social network can be defined as a set of relationships between objects (or nodes) [11]. Nodes can be people, entities, organizations or even all of them. The relationship among entities can be any connection they may have in common, e.g., an edge can link people who work in the same department, or developers who work in the same file.

Social networks are a way to understand and analyze interactions and the social organization of a group. Social Network Analysis (SNA) is the application of mathematical techniques to study social networks. Note that SNA does not focus in the attributes of the nodes, but instead in their relationships (the edges) [14].

A social network can be defined according to some structural and topological properties. Some structural properties include: node degree, weighted degree of a node, distance centrality of the node, proximity degree, and betweenness centrality [12].Topological properties include: density of the network, distribution degree, network diameter, and finally, cluster degree.

Social networks can be broadly classified in two-types: *1-mode networks*, which represent relationships between social entities of the same type, for example, who is friend of who, who asks for advice from who, who depends on who; and *2-mode networks*, which represent relationships between different social entities, for example, people who attended a meeting, developers who fixed a given bug. It is important to note here that from a 2-mode network, it is possible to geta 1-mode network through mathematical operations in the matrix [13]. For instance, if we have a developer vs. bug matrix (2-mode) we can obtain a developer vs. developer matrix (1-mode) through matrix operations.

4.2 The PMD Project

In order to validate the ideas presented in this paper, we conducted a case study with a free/open source project. The project analyzed is the PMD [21]: a project in development since 2002. This project aims to provide tools for Java source code analysis. It finds unused variables, empty catch blocks, unnecessary object creation, and so forth.

We extracted data to create and analyze four different social networks for this project:

1. The social network for the *open discussion* forum, which contains threads of general discussion;
2. The social network for the *developers* discussion list, the threads started and used by developers to discuss issues regarding the construction of the PMD project;
3. The social network extracted from the discussions about the bugs; and
4. The social network extracted from the source code.

The interval of analysis for the first three networks was from 2003/02/24 to 2003/11/03. These networks were created using the OSSNetwork tool [18]. The source code of the PMD project was collected starting on 2002/06/24 as we will explain later.The social network for the source code was created using the Transflow tool. Both tools, OSSNetwork and Transflow are described in the following subsections.

4.3 OSSNetwork

OSSNetwork is a tool that extracts information from different FLOSS repositories to create social networks. Extracted information includes discussion about bugs, mailing lists, forums, etc. Information is extracted by parsing HTML information available in SourceForge, RubyForge or Codeplex repositories. OSSNetwork also parsers mailing lists information from the Apache community web-site and IRC data from ircbrowse.com. After the parsing, OSSNetwork allows its users to visualize the extracted information through social network graphs (sociograms) and to analyze these networks with SNA metrics. While the parsing is performed on HTML, the visualization subsystem is based in the JUNG framework [15]. The social networks generated by the environment can be exported to files in CSV and DL formats to be used in other social network analysis tools like UCINet [16].

In short, the OSSNetwork environment allows one to (i) retrieve information from FLOSS repositories, (ii) store this information in a database, (iii) generate different social networks from this information, and, finally, and (iv) analyze these networks using tools to manipulate, edit, and execute algorithms. The OSSNetwork aims to minimize the effort of researchers interested in the study of FLOSS communities. This is done using modern software engineering techniques that allow one to easily add new FLOSS repositories, ways of creating social networks and additional SNA metrics. More details can be found in [18].

4.4 TransFlow

TransFlow is a plugin for the Eclipse IDE (Integrated Development Environment) which allows the collection of data from configuration management repositories of both FLOSS or commercial projects. Its goal is to understand the evolution of software developers' source code contribution to open source projects. Accordingly, it is necessary to extract information from configuration management repositories like SVN and CVS. Transflow calculates metrics regarding these contributions using different approaches. For the goal of this paper, Transflow performs a co-changes analysis of the source code modification history and creates a matrix where software components are connected by taking into account the frequency that they have been changed together (i.e., in the same check-in): if two files have been changed together in the same check-in, an edge is created to link these two files. This approach is based on Zimmermann and colleagues [19]. In other words, by taking into account the history of the software project, it is possible to create a matrix that describes the dependencies between files that arise out of the changes being made to these files. The values in the cells of the matrix indicate the dependencies that a file in a particular row has for each file in a particular column. Once this matrix is created, this means that project data has been collected and that dependencies have been calculated. Then, a XML file is generated by the tool in order to create different visualizations of the history of the project activity.

For this paper, we are interested in a scatter-plot visualization: Cartesian coordinates are used to display values for two variables in a given dataset. An example of such visualization generated by TransFlow for the PMD project is shown in Figure 2. Each square represents a check-in that has happened in the source code repository. Colors of the squares are used to identify authors who performed the commits. The X-axis can be used to describe either the number associated to the check-in or the date in which it happened, while the Y-axis can describe either the average centrality or the maximum centrality of the files modified in a check-in. In Figure 2, the Y-axis describes the average centrality of all files changed by a particular commit, while in Figure 3 only the maximum centrality is presented. The visualization shown in Figure 3 was created using a function of the tool that enables one to group commits of the same author. In this particular case, commits are grouped in groups of five.

The goal of the visualizations presented in Figures 2 and 3 is to investigate how open source software developers' check-ins become more and more complex over time: the centrality of the changes (represented in the Y-axis) made by open source developers increased over time (X-axis), indicating how these developers become more and more central for the project.

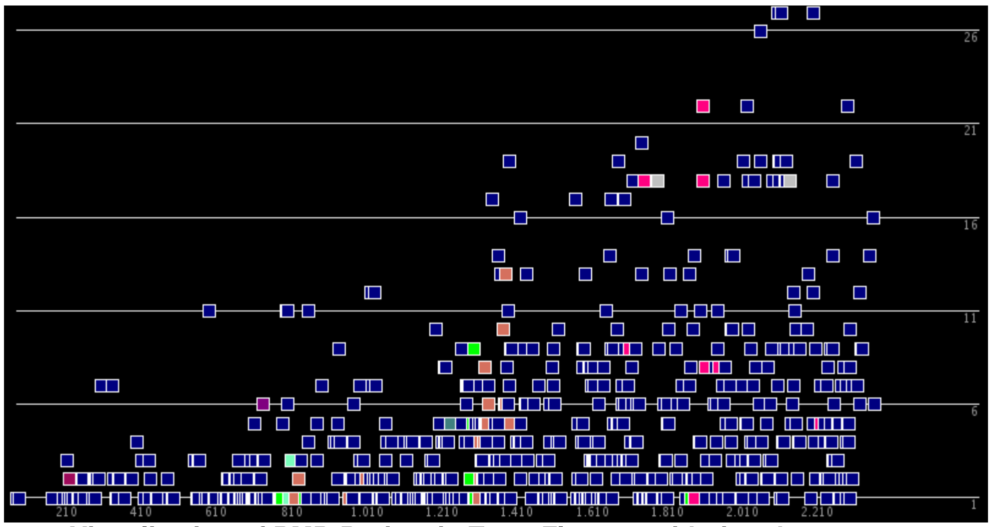

Figure 2 - Visualization of PMD Project in TransFlow considering the average centrality.

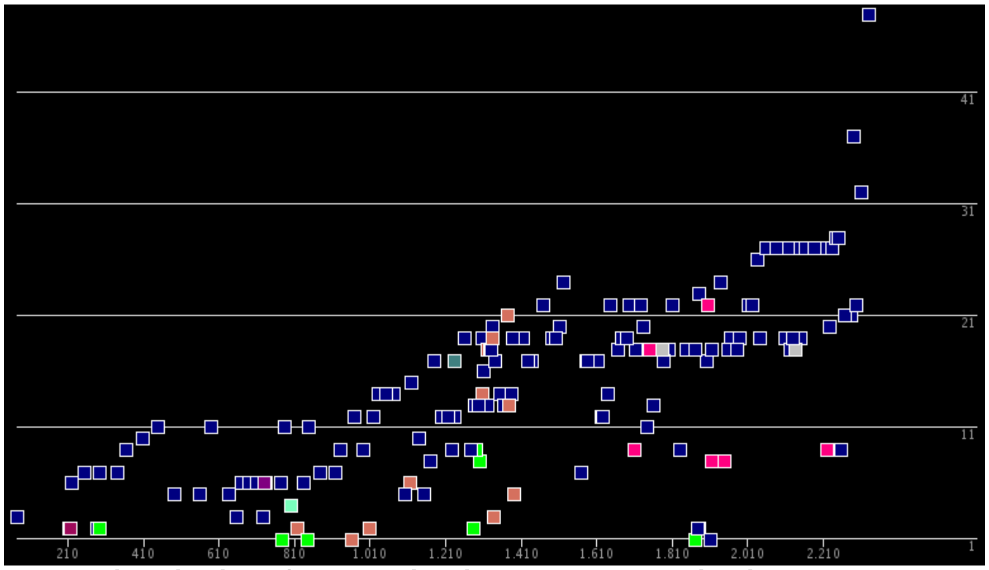

Figure 3 - Visualization of PMD Project in TransFlow considering the max centrality and grouping the commits.

Since TransFlow generates matrices with dependency information about the project files and has access to authorship information (who changed each artifact) from the configuration management repository, it is possible to combine these two types of information to create a social network that indicates dependencies among software developers [20]. These social networks are the ones used in the analysis described in this paper.

Note that in order to create a social network of a given date using this approach, we need the commits or check-ins that were performed before that given date, otherwise, there would not be no social network. That is the reason why the source code of the PMD project was collected starting on 2002/06/24 instead of 2003/02/24 as in the other social networks (mailing lists and bugs).

4.5 Social network creation

To initiate our analysis, we extracted data from the forums of (i) open discussion and (ii) developer discussion, (iii) bug track system, and (iv) source-code from the configuration management system of this project. Social networks were created for each one of these datasets.

The social networks have been constructed in the following way. First of all, we extracted information (message in a forum thread, class created, message related to a given bug and so forth) for each developer who produced it, generating a developer vs. information matrix. One matrix was created for each social network. This matrix is a two-mode matrix. After that, this matrix is multiplied by its transpose matrix, resulting in a one-mode matrix of developer vs. developer [13]. All non-zero values are considered as a edge in the social network.

After the social networks were created, we calculated different metrics for each one of these social networks. To be more precise, information centrality was calculated for the bugs and forums networks, while the betweeness centrality was calculated for the source code network. Each metric was visualized using Sargas, so that a star represented each actor of these networks and each face of the start displayed the metric for that actor in the corresponding social network. Whenever an actor did not participate in a particular social network, the face of the star corresponding to that social network would not be drawn. Visual inspection of the star was then used to identify distinct groups of actors based on patterns of action and interaction (depending on their metrics) in the social networks. To be more precise, we identified six different groups that are discussed in the following section alongside the illustrative Sargas representation for that group.

5. Results

As mentioned before, we identified six different groups that are discussed below. For our analysis of the PMD project, the colors of each network are as follow: an open discussion network is indicated by a yellow face (face pointing south), bugs network are indicated with white (face pointing north), the source code network is filled with pink (face pointing east) and the developers' discussion mailing list network is indicated with the red color (face pointing west).

A) The first group represents actors who are significantly present in three or four social networks at the same time. Our interpretation is that these actors are "brokers" using the social network terminology [13]: because they actively participate in different social networks, they are responsible for the flow of information among them. For instance, they provide information about users' needs to "core developers" or implement these needs themselves [17].

We identified three actors that are present in all four networks and two more actors who are present in three social networks (developers, open discussion and bugs networks). Their representation is depicted in Figure 4. This group is represented by contributors user 1, user 2, user 3, user 4 and user 5. Within this group, we can find the team manager, user 1.

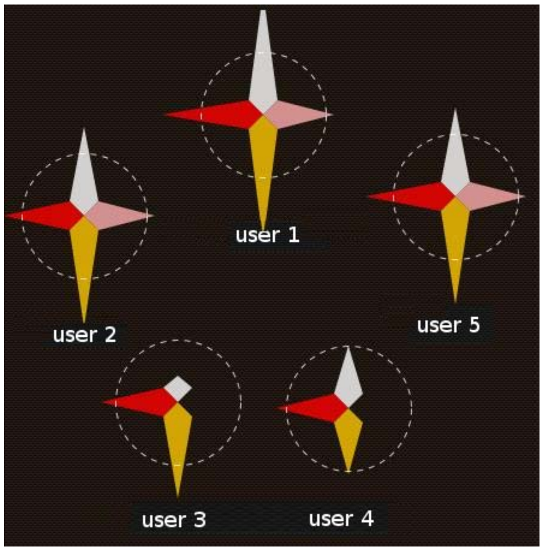

Figure 4 – Group A: Brokers who exchange information among different social networks.

B) The second group that we identified represents members of the project who are both in the open discussion and bugs networks. Figure 5 displays this group. Developers in this group have a singular pattern: they have the white (north) and the yellow (south) faces of the star. In this case, our interpretation of the data is that these members are responsible for "translating" possible problems identified by PMD users in the open discussion forum into actual bug reports that are discussed in the bug discussion network. As such, they perform an important role to the success of the open source project.

C) The third group represents actors who are active in both developers' and open discussion networks. There are six developers in that category. In their Sargas' representation displayed in Figure 6, they have the red (west) and yellow (south) faces of the star. In this Figure it is easy to observe that these actors are central (highly active) in both networks.

A possible interpretation of this pattern of action and interaction in open source communities is that these developers perform the role of "user proxies", i.e., they collect users' needs discussed in the open discussion forum and report them in the software developers' forum.

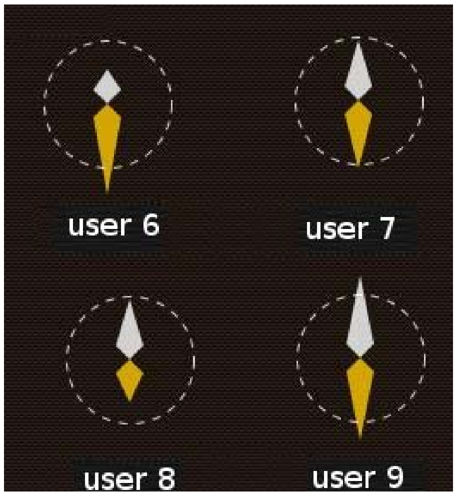

Figure 5 – Group B: users who contribute with bug detection and submission

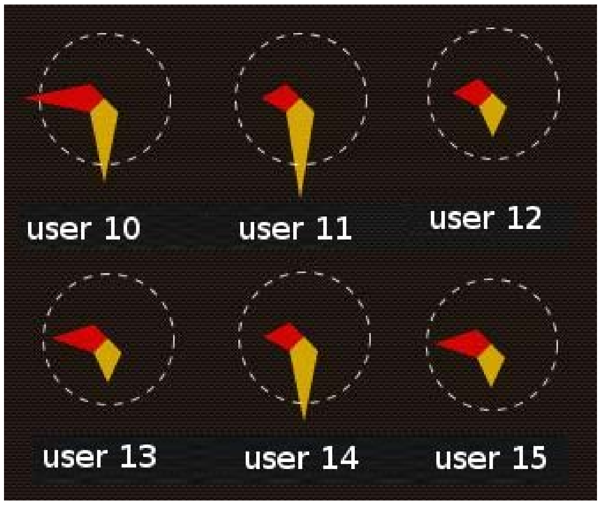

Figure 6 – Group C: User proxies.

D) The fourth group that we identified represents the set of actors who are somewhat central in the developers and bugs networks, but who are not present in the source code network. That is to say, these developers participate in all the discussion about the open source project that occur in the PMD developers forum and in the bugs networks. However, these developers do not actively contribute source code to the PMD project. This group is displayed in Figure 7. There are only two developers in this group: user 16 and user 17. Note that user 16's centrality in the source code network was very small (as indicated by the small pink face (east) of his corresponding star) that we decided to assign this developer in this category.

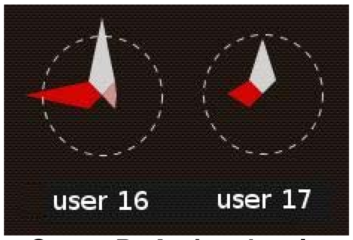

Figure 7 – Group D: Active developers in the technical discussion about the project, but who do not make major changes in the source code.

There are two alternative explanations for the work performed by these developers. First, these developers were core developers [13] in the past, but now they are not part of the core anymore. Second, those might be developers who are engaging more and more in the discussion about the source code, but who were not yet awarded commit privileges.

In order to eliminate these alternative explanations, we used Transflow to get information about contributions of these developers in two periods: *before* and *after* the period analyzed in this paper. We generated Transflow visualizations for these two developers for these two periods. These visualizations are similar to the ones described in Figure 2 and 3, and they indicated whether these developers had committed changes to the PMD repository in these periods and, if that is the case, the evolution of the complexity of these commits. Regarding user 17, we found no evidence about him being a committer earlier in the project, nor becoming a committer later. However, we observed that user 16 has indeed made contributions to the source code of the PMD project *after* our period of analysis. His contributions to the project were somewhat limited and included changes in the documentation, credits and paths. Although one might argue that these changes were not source code per se, we argue that these changes are still relevant to the success of the open source project.

E) There is only one actor who is only in the source network and the open discussion at the same time: his login is user 18 and he defines the fifth category that we identified.

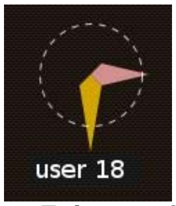

Figure 8 – Group E: Interaction with users.

F) Finally, there are different groups of project members who are different from all other groups: these actors participate in a single social network at a time. For instance, there are members who only participate in the source code network, others who only participate in the open discussion forum (and consequently social network) and the same is true for the developers' forum and bug discussions. These groups are all represented in Sargas in Figure 9.

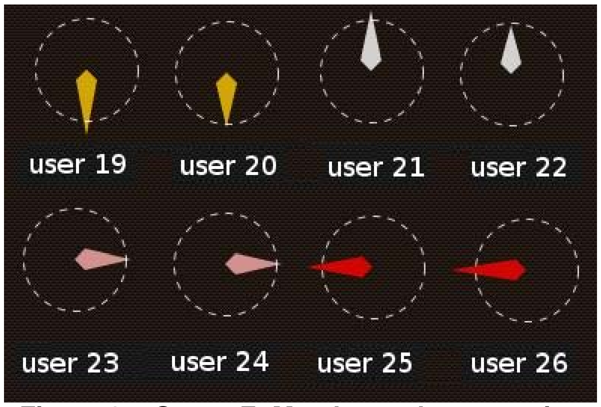

Figure 9 – Group F: Members who are active in only one network at a time.

In short, there are 110 actors. The brokers group contains 5 actors. The bug detection group contains 4 actors. There are also 6 user proxies, 2 developers in the fifth group and 1 actor in interaction with users group. The isolated actors group has 50 actors in open discussion's list, 15 actors in the developers' forum, 13 actors in the bugs discussion network and finally, there are 13 developers in the source-code network.

6. Discussion

The so-called "onion model" of participation has been used to categorize the members of a FLOSS project, i.e., despite the freedom of these projects; there is some structure in the open source teams [17]. In this model, open source members can be classified in only one of two groups: the user group and the developer group. In the *developer group*, there are the *core actors*, who are responsible for contributing code and the decision-making in the project. Core developers also manage CVS releases and coordinate peripheral and central developers. This group is generally small. The second group of actors is the *co-developers* or *central developers*. This group is more numerous than the core developers and they contribute with bug fixes, or submit patches, revisions, etc. Their contributions are evaluated by the core developers. They can also offer support and write documents. The third group is the *peripheral developers*: they fix bugs (but not in a regular basis) and submit contributions. Finally, there is the *project leader*, who has the project vision and gives the directions that should be followed. In the *user groups*, there are active users and non-active users, or passive users. Active users report bugs and exchange information about them in the discussion lists. They are important because this interaction with developers is healthy to the open source project. Passive users, on the other hand, have no interaction with the development team. They just use the software for their needs.

One problem with the "onion model" of open source participation is that it classifies participants in only one of two groups: the user group and the developer group so that developers who perform activities in the two groups are no easily classified. We argue that by using multiple social network analysis as we describe in this paper, it is possible to detect nuances and details of open source participation that are more expressive than the "onion model". Accordingly, we describe how the groups that we identified in the previous section can be, to some extent, mapped to the onion model, and furthermore, how some of our groups are actual intersections between roles in the onion model. For instance, group A, the brokers, are members with high centralities values who translate knowledge from users (in the open discussion forum) to the core, active developers, or active users who collaborate with bug submissions. The group B represents the active users according to the onion model, who are responsible for reporting bugs, but in addition, we argue that these members also identify bugs because they participate in the open discussion forum.

The group of members called group C is very important to the PMD project because it can be understood as "user proxies", that is, these members act as bridges between users and developers. Note that this particular role is not envisioned by the onion model. In contrast, the group D is more or less equivalent to peripheral developers in the onion model.

The group E had only one actor that we assumed was a developer where his function is to be the interaction with the community.

Finally, open source members generally classified in group F represent actors who are active in one social network only. Accordingly, team members who participate in the open discussion forum are "equivalent" to passive users in the onion model.

7. Conclusion and Future Work

Several authors have performed social network analysis of open source projects. These analyses are insightful and have served to increase our knowledge of how these projects work and coordinate themselves. In this paper, we go further by performing analysis of *multiple* social networks at the same time. We described our approach, based on a (Starplot) visualization of social network metrics, as well as, our tool, Sargas, that can be used to perform this analysis. We illustrated our approach with a case study of the PMD project and discussed how our results can be used to extend the so-called "onion model" open source participation.

An obvious limitation of our approach, and of all previous approaches based on social network analysis of open source projects, is that we are not analyzing the content of the messages exchanged among open source members. We are solely focusing on the structure of the resulting social network. We plan to perform additional analysis of other open source projects as well as to improve our approach and tool to facilitate this analysis.

Acknowledgements

This research was supported by the Brazilian Government under grant CNPq 479206/2006-6, by the Universidade Federal do Pará through the PARD and PIBIC programs, and by a Microsoft grant.

8. References

[1] Wagstrom, P., Herbsleb J. and Carley K. A Social Network Approach to Free/Open Source Software Simulation, pp 1, 2005.

[2] Lopez-Fernandez, Luiz, Robles, Gregorio, Gonzalez-Barahona, Jesus M. Applying Social Network Analysis to the Information in CVS Repositories. GSyC, Universidad Rey Juan Carlos, 2004.

[3] Gloor, P. Swarm Creativity - Competitive Advantage through Collaborative Innovation Networks. Oxford University Press, New York, January 2006.

[4] de Souza, C., Froehlich, J. and Dourish, P. Seeking the Source: Software Source Code as a Social and Technical Artifact. Sanibel Island, FL: s.n., ACM Conference on Group Work, 2005.

[5] Jensen, C. and Scacchi, W. A Reference Model for Discovering Open Source Software Processes. Limerick, IR. Third IFIP International Conference on Open Source Systems, 2007.

[6] Halloran, T. and Scherlis, W. High Quality and Open Source Software Practices. Orlando, FL: s.n., Workshop on Open Source Software Engineering, 2002.

[7] Ducheneaut, N. The Reproduction of Open Source Software Programming Communities. Berkeley, CA: UC Berkeley, School of Information Management and Systems, 2002.

[8] Jensen, C. and Scacchi, W. Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. International Conference Software Engineering. 2007.

[9] Gao, Y. and Madey, G. Network Analysis of the SourceForge.net Community. Limerick, Ireland: s.n., International Conference on Open Source Systems, 2007.

[10] Chambers, Cleveland, Kleiner, and Tukey, Wadsworth. Graphical Methods for Data Analysis, 1983 (pp. 160-161).

[11] Kadushin, C. Introduction to Social Network. February 17, 2004.

[12] Stephenson, Karen and Marvin Zelen. Rethinking Centrality: Methods and Examples. In Social Networks 11. (North-Holland: Elsevier Science Publishers B.V., 1989) pp. 1-37.

[13] Wasserman, S. and Faust, K. Social Network Analysis: Methods and Applications. Cambridge, UK and New York: Cambridge UniversityPress, 1997.

[14] Hanneman, R. and Riddle, M. Introduction to Social Network Methods. Riverside, CA: University of California, 2005.

[15] Fisher, D., et al. Analysis and Visualization of Network Data Using JUNG. Java Universal Nerwork/Graph Framework. [Online] http://jung.sourceforge.net/doc/JUNG_journal .pdf

[16] Borgatti, S., Everett, M. and Freeman, L. UCINET 6 Social Network Analysis Software. Analytic Technologies -- Social Network Analysis& Cultural DomainAnalysis. 2006.

[17] Crowston, K. and Howison, J. The Social Structure of Open Source Software Development Teams. Seattle, WA: s.n., International Conference on Information Systems, 2003.

[18] Balieiro, M. A.; Sousa Jr, S. F.; de Souza, C. R. B. Facilitating Social Network Studies of

FLOSS using the OSSNetwork. In: International Conference on Open Source Systems, 2008, Milan. Proceedings of International Conference on Open Source Systems, 2008.

[19] Zimmermann, T., P. Weibgerber, et al. (2005). "Mining Version Histories to Guide Software Changes." IEEE Transactions on Software Engineering 31(6): 429-445.

[20] Cataldo, M., P. A. Wagstrom, et al. (2006). Identification of Coordination Requirements: implications for the Design of Collaboration and Awareness Tools. 20th Conference on Computer Supported Cooperative Work. Banff, Alberta, Canada, ACM Press.

[21] PMD Project. Available in http://pmd.sourceforge.net. Access on September 17.