Using Social Network Analysis to Inform Management of Open Source Software Development

Sanjeev Kumar
Lubar School of Business, University of Wisconsin Milwaukee
kumars@uwm.edu

Abstract

The community-based open source software (OSS) development model has emerged as a viable alternative to firm-based traditional software development. The naturally evolving structure of collaborative relationships among software developers is a major distinction between the OSS development model and the traditional development model. Conventional statistical methods that focus on individual cases and their attributes cannot properly inform the management of the naturally evolving collaborative relationships in open source projects. We emphasize social network analysis as a method especially suitable for management of open source development projects, because it focuses on relations among individuals rather than attributes of individual cases. We show how open source development can be represented as a collaboration network graph and how the network can be characterized by various network structure metrics. We present four metrics as a starting point – size, centralization, density and clustering; that are most useful in revealing collaborative relationships in OSS development process. We discuss how to generate collaboration network for OSS projects and how to calculate the metrics. We further describe how these metrics can assist in effective management of open source software development process. We conclude by presenting preliminary empirical evidence in support of the metrics.

“Any piece of software reflects the organizational structure that produced it.”

-- Conway’s Law [1]

1. Introduction

Open source software (“OSS”) development projects are Internet-based communities of software developers who voluntarily collaborate to develop software that they or their organizations need and provide free access to the software source code [2]. OSS has achieved significant success with several popular products such as Linux, Apache and FireFox. OSS is attracting significant investment from large commercial firms and a large number of software projects are under development under the OSS umbrella.

The community-based OSS development model has emerged as a viable alternative to firm-based traditional software development. An OSS development community consists of self-organized software developers who voluntarily contribute to an open source software product by writing code for the project or doing other tasks such as testing, and documentation. The naturally evolving structure of collaborative relationships among software developers is a major distinction between the OSS development model and the traditional development model that assigns tasks and prescribes predetermined collaborative relationships between developers [3]. Studies have shown that the self-organized nature of collaboration in OSS development has a significant impact on the development process and its outcomes [4, 5]. In order to leverage the strength of the OSS process, it is critical for OSS management to understand the structure and impacts of the naturally evolving collaborative relationships among OSS developers [6, 7].

Examining the collaborative relationships has not been a focus of conventional software management techniques, because commercial software development is usually based on pre-determined collaboration structure among developers. In contrast, OSS emerges out of unstructured environment and retains a decentralized, informal and cooperative nature [8]. Techniques and tools used in commercial software development may not be effective in managing an OSS development process, because they tend to focus on individual cases and their attributes rather than relationships among individuals. Although current OSS development process has developed several informal management and control tools including the mailing lists, online bug tracking
software, the code repositories like CVS (Concurrent Versioning System) database and various kinds of discussion forums, these tools are not sufficient to sustain OSS and increase its scale of operations to serve as a viable alternative to traditional software development. The OSS development process needs to adopt management and control methods that are more formal, robust and are capable of providing OSS project leaders more concrete measures of the health of the OSS projects. OSS projects have historically suffered from chronic delays and high failure rates [9]. There has been concerted effort recently to improve management of open source development projects [10-12]. This research attempts to contribute in that direction by emphasizing techniques especially suitable for management of the naturally evolving collaborative relationships in OSS development.

Attempts at better management of OSS development projects have a long history. Studies that show that project leaders can have significant impacts on OSS project outcomes. They can influence behaviors of project participants, code contributions and other aspects of the project [8]. For example, in a meta-study of OSS case studies, Gallivan concluded that OSS projects can be managed through various mechanisms and “ensuring control is an important criterion for effective performance within OSS projects” [13]. Hence, even in OSS context, project administrators can leverage suitable management and control methods to improve the OSS development process.

2. Social Network Analysis

Social network analysis is the study of network structures made by nodes (e.g. individuals or organizations) that are linked to other nodes through some kind of relationship (e.g. transaction, collaboration, family relations). Social network theory models actors as nodes of a graph joined by their relationships depicted as edges [14]. One significant difference between conventional statistical methods and social network analysis is that the former focuses on actors and attributes whereas the latter focuses on actors and relations. The social network approach has become a key technique in a variety of disciplines such as sociology, anthropology, information science, biology and economics.

To illustrate social network theory, let us consider the context of collaboration through e-mail in a virtual team. Let us assume that the team has 11 members and they write e-mails to each other when they wish to collaborate with a teammate (e.g. ask questions, get advice etc.). In social network terminology, each employee is a node and any instance of e-mail between two employees is a link between them. The network shown below can graphically show what have transpired over the projects – how has the collaboration happened. In figure 1, the dots represent team members (i.e., nodes), identified as A through K, and the links between the dots represent instances of e-mail between the given two people.

**Figure 1: Example of a Social Network**

Figure 1 shows that person D was in e-mail conversations with A, B, C, E and F; H collaborated with G and I and so on. The graph also shows that person K did not have a collaborative relationship with anybody and hence does not have any links attach to it.

The network shown in figure 1 can be improved to include more information. We can extend the basic network shown in Figure 1 to include information about direction of interaction (by assigning links a specific direction) and the volume of interaction (by assigning links a weight, heavier links for more interaction).

Once we depict a particular context as the appropriate network graph, the social network theory provides well developed metrics to study various aspects of the network [14]. In the following sections we first show how OSS development can be mapped onto a graph and then we discuss the network metrics relevant to OSS development process management. Social network analysis can be used in the OSS development context at the inter-project level or intra-project level. Previous studies have used inter-project social networks successfully in OSS context [15, 16]. However, as the focus of this study is on management of individual OSS development projects, we will conduct the analysis at a project...
level with individual developers as nodes and their collaboration as edges in the network graph.

**Social Network Analysis of Open Source Development**

Social Network Analysis is a widely studied area with significant amount of extant relevant research (e.g. [41]). Previous research has successfully applied Social Network Analysis in the OSS context. Collaboration networks like the one developed later in this section have been extensively used before [37]. Social Network Analysis was used to create affiliation network for large OSS projects and to calculate network measures [38]. Using Project Networks and Developer Networks, OSS development was shown to be scale free with power law distribution and small world characteristics [39]. In this research, we extend the previous research to focus on four key network measures and show that they can be used to effectively manage OSS projects.

The OSS development process can be well represented by using a collaboration network [17, 18]. In collaboration network of OSS development, individual developers are the nodes and instances of collaboration between two developers form the link between them. Since the amount of collaboration between two developers can vary, it would be appropriate to use weighted links to depict an OSS collaboration network. One such collaboration network generated for a real OSS project is shown in Figure 2 below.

![Figure 2: Collaboration network of an OSS project](image)

Collaboration network for any OSS project can be drawn using information available in the code repository of the project. Code repositories of OSS projects store every change to the source code, including the authorship information, making them a rich source of information about the project’s health [19, 20]. These code repositories are usually organized in a tree structure with the project as the base and project modules or parts as branches. In any module, sub-modules are represented as further branches and so on. At the bottom of the tree lie individual source code files. Each file represents a programming task or function. The code repositories keep a complete version history of these files.

For the purpose of illustrating the process of developing the collaboration network, we consider contributing to the same code file in the OSS code repository as an instance of collaboration between two developers. Such operationalization of collaboration has been successfully used in previous research [21, 22]. As code repositories are quite large in size and the process of identifying collaboration instances, as per the definition above, quite simple and standard; the task can be easily automated. The authors used a simple script to parse code repository information and identify collaboration instances for the collaboration network shown in figure 2. Once collaboration instances have been identified, the drawing of the network and calculation of different metrics (discussed later in the article) can be left to several specialized software packages available including many free and open source ones. We have used the UCINet software to draw figure 2 and to calculate various network metrics used later in this paper [23]. Several research papers have followed similar methods as described above to generate social networks of the OSS projects (e.g. [17, 22]).

**Metrics for OSS Collaboration Network Structure**

The collaboration network generated from the code repository of an OSS development project can be quantitatively characterized by a number of metrics. However, metrics that are likely to be most useful in managing and monitoring an OSS project are those that capture the unique attributes of the OSS development process. OSS development process involves an informal and organic collaboration approach, wide variance in team sizes and a lack of strict hierarchical and central control [3, 5]. These unique aspects of OSS development process are captured by the following four metrics: network size, network centralization, network density and network clustering. We discuss each of these metrics, how they relate to unique aspects of OSS development and their likely role in managing the OSS projects one by one below.
Network Size: Network size shows the scale of collaboration. It is defined as the number of nodes in the collaboration network [14]. This simple measure points to the most essential distinction between open source and traditional software development process. Open source development is a voluntary process and relies on contributions from a large group of distributed developers. The first measure of success of an OSS project is the number of active developers it manages to attract. Most of the OSS projects started at SourceForge.com fail to attract any developer other than the person who started the project and hence cannot get enough code contributions to make progress towards completion. Hence, network size is an effective indicator of success of the OSS project in attracting developers.

Network Centralization: Network centralization measure indicates the extent to which a network is centralized around one node or a group of nodes [14]. The more centralized networks resemble a star structure or a hub-like structure while the less centralized networks resemble a chain like structure. Centralization of a network is measured in comparison to the most central network – the star network. In the star network, one central node connects to all of the other nodes while all other nodes are only connected to the central node. Any deviation from this structure indicates a reduction in group centralization. The network centralization measure varied from 0 (for a chain network) to 1 (for a star network). Network centralization measure captures the informal and organic aspect of OSS development where individual developers work with much lower central hierarchical control compared to traditional software development.

Network Density: Network density reflects the collaboration frequency of the network – how dense is the collaboration network. It is measured as the ratio of actual collaboration links to all possible links in a network [14]. Similar to network centralization, network density is a relative measure that compares the frequency of collaboration in a given network with the maximum collaboration possible i.e. a network where all nodes are connected with every other node. Accordingly, network density varies from 0 to 1. The network density measure captures the informal collaboration aspect of OSS development process. In contrast to traditional software development, collaboration is not mandated and emerges organically out of developer interactions so network density can also be seen as a measure of group cohesiveness.

Network Clusterness: Network clusterness indicates whether the nodes are connected primarily with a subgroup of teammates or evenly with all the team members. Network clusterness or the clustering coefficient for the network is measured as the average density of each node’s neighborhood (i.e., the graph made of all the nodes directly connected to the ego node) [14]. A low clusterness value indicates that developers are working evenly with the whole team, while a high clusterness value means that the team has different clusters of developers that work primarily within their own clusters. While the network density measure discussed before shows the level of collaboration in the network, the clusterness measure provides further information about the texture of collaboration – whether the collaboration is with the whole team or primarily with a smaller cluster of developers.

The four metrics above capture the major characteristics of structure of the OSS collaboration network. They reflect unique aspects (e.g., variance in developer team size, lower central control and organic collaboration) of the OSS development process compared to the traditional software development process.

3. Using Network Metrics to Inform Management of OSS Projects

While the network metrics discussed above have been widely used in different context, we will explore how they can provide information about different aspects of the collaborative relationships in open source development process. These metrics can signal specific problems and then the project leaders can take remedial actions to effectively manage the development process. We again look at the four metrics to discuss how these can inform the management of the OSS development project.

Network Size: A small network size relative to the size of the software being developed may imply that the project does not have enough human resource to make satisfactory progress. In such cases, project leaders need to actively attract developers to the project. On the other hand, too many developers can also be a problem, because communication and coordination cost can increase exponentially with the size of the network [24]. Brooks’ Law in traditional software development indicates that for complex development activities, small surgical teams (also known as chief programmer teams) are more suitable than large and loosely coordinated teams [25, 26]. In fact, most successful OSS projects have a small and
dedicated core group of developers. However, if the programming tasks have little interdependency and do not require frequent interventions by project leaders, a large number of developers can be an asset as they can gain higher productivity from effective division of labor [27].

**Network Centralization:** A high degree of network centralization indicates that the development effort is highly concentrated with one or few core developers. This can be a positive feature if the project requires consolidation of multiple knowledge domains [28]. The core developers can serve as the hub for knowledge exchange among developers. In complex projects where significant coordination is required, a more centralized team may work better as it has lower communication and coordination overheads [25]. On the other hand, high centralization indicates that the development process relies on a few core developers. They can become the vulnerable points of the network. Departure of core developers can break down the collaborative relationships among other developers. Cognitive overload on such central core developers can make them the bottleneck of the development progress. Low centralization reduces the vulnerability of an OSS project to the departure or cognitive limit of a few core developers. Although low centralization network is less effective in consolidating knowledge domains in complex software project, it can be optimal when programming tasks have lower interdependency and do not require frequent knowledge exchange among developers. In contrast to the network size measure discussed above, centralization is considered scale independent and hence is not normalized by the project size.

**Network Density:** Network density measures the extent of collaboration in the development team. Collaboration in OSS teams emerges organically and hence a higher degree of collaboration, indicated by a higher network density value, shows that the team works well together and leverages skills available with each other. However, high network density may impose significant coordination costs that may negatively affect the success of the project. A high density network facilitates the knowledge sharing among developers through wide scale collaboration among developers, but it also increase communication and coordination needs of the projects. This may not be suitable for complex projects that already impose high coordination needs or in projects that have limited central leadership resources available to coordinate and manage the high degree of collaboration. On the other hand, if the coordination and communication needs are not a problem then a low density environment may not be using all the available skills and resources available in the developer group because individual developers prefer to work alone and not collaborate with other developers who may have complementary skills.

**Network Clusterness:** Network clusterness indicates whether the developers are working in one large uniformly collaborative group or they have divided themselves into different virtual clusters of developers with developers collaborating more within their own cluster than with developers in other clusters. It would be more effective for the clusterness of the collaboration network to be aligned with the clusterness of the development tasks. If the development requirements include different groups of tasks with different skills and resources needed, then a corresponding clustering of the development team may bring developers with similar interests and objectives together and make them more effective. However, if the collaboration network is more clustered than what is required by the development tasks, then it may signal that the team has different groups that are not comfortable working with each other. Such high clusterness in collaboration network can lead to lower levels of integration among modules or slower development due to lower knowledge sharing and collaboration.

While the current level of collaboration network metrics can provide important insights into the health of the OSS development project, project leaders can also monitor changes in the metrics over time to check how the project is evolving. For example – if clusterness of the network is increasing significantly over time, it may indicate a growing risk of forking; that is, different subgroups of developers of the overall team continue to drift apart and not work outside their own subgroups. Similarly, increasing centralization can indicate that the development effort is getting concentrated on a few central hands with a corresponding risk to the project because of the departure or cognitive overload of the central developers. An increase in centralization with a decrease in density may indicate that not only a few developers are controlling most of the contributions but they are collaborating less and less with other developers. Thus, OSS development project leaders need to monitor not just the current level of network metrics but also how they are changing with time to make sure that the development project is healthy and progressing smoothly.

**4. Proof of Concept**
To show that the metrics presented in the paper are in fact effective as management tools for managing an OSS development project, it is essential that we show that the metrics have a significant impact on the OSS development process. We have developed a proof of concept for the purpose using data from code repositories of OSS projects from SourceForge.net. Due to limitations of submission size in this paper, this section is necessarily brief and provides only essential details to establish the metrics as valid and effective in influencing the OSS development process.

From hundreds of thousands of OSS projects hosted at SourceForge.net, we selected those that (1) keep detailed CVS information in the public domain, (2) started between Feb 2001 and Nov 2001, and (3) have at least 10 developers in the collaboration network. The first criterion makes our data collection practically possible. The second ensures that the sampled projects have sufficient elapsed time, so that periodic surge or dip in development activities will not significantly bias the construct measures. The proximate starting times for the projects in the sample also helps to eliminate longitudinal effects that may confound our results. The third criterion is to safeguard the integrity of the collaboration network measures. Collaboration network measures are sensitive to the size of the network [14]. Particularly, when the network size is small, some network measures, such as centralization, become meaningless. Therefore, we have restricted our sample to projects with at least 10 developers in the collaboration network. This resulted in a sample size of 96 projects.

To show that the four metrics discussed above are effective in influencing the OSS development process, we need to develop a good proxy for the project development success. Further, we need to include elements of the characteristics of the development effort (for example – the complexity of the development effort) in the analysis as well. Finally, our analysis must control for various factors associated with the OSS development project such as the programming language, OSS license type and the project development stage.

We choose Project Growth Rate as the proxy for the success of OSS development process. Project growth rate is measured by lines of source code added per developer per day to an OSS project. Software development literature provides for several well studied success measures including some specific to OSS development [40]. We chose Project Growth Rate for this research as it is one of widely accepted success measures that can be easily measured and followed. While previous research has identified problems in using such as success measure, we consider it adequate for proof of concept analysis in this paper.

To characterize the complexity of the OSS development project, we have attempted to develop a proxy the commonly used construct of Software Structural Interdependency. Conceptually, software structural interdependency is measured by “the strength of association established by a connection from one module to another.” [29]. A direct operationalization of this construct would require a manual review of software source code and evaluation of its interdependency. Although automatic tools (e.g., Lattix) are available for evaluating software architecture, these tools were found infeasible for the overwhelming amount of source code and the wide range of programming languages in our sample. Hence, we have taken a more tractable approach of developing a surrogate measure, number of source code task files in the code repository, to measure software structural interdependency.

It can be argued that interdependent processing elements are likely to be grouped into the same task file in the code repository so that developers can conveniently work on the interdependent processing elements by checking out one file. For a given number of processing elements, the more interdependent they are, the more they are likely to be arranged into same files, leading to fewer task files. This structure is likely to evolve as each successive contribution from one or more developers concentrates similar programing elements together in one task file.

As an example, consider two projects, A and B, both have 100 processing elements (e.g., functions or procedures). Project A has a high level of structural interdependency and hence any modification to an element in A would invoke related changes in an average of nine other elements. To ease the task of developers, these related change sets are likely to be grouped into the same task file, which in turn results in around ten task files in project A’s code repository. In project B, processing elements have low interdependency level. A change to an element in B only involves related changes in three other elements. When related change sets are grouped into the same task file, project B may result in around 25 task files.
in Project B’s code repository. In general, lower numbers of task files represent higher levels of software structural interdependency and vice versa. We normalized this measure by product size (measured by total lines of code of a project) and number of code modules (represented by last folders in the code repository file tree) to ensure that the structural interdependency measure is not biased by the code repository tree structure and product size.

We controlled for the following variables based on previous literature:

**Product Size:** In software engineering literature, product size has been identified as an important factor in manpower, time and productivity of a project [30, 31].

**Programming Language:** In our data, Java, C++ and C are the most frequently used. We created three dummy variables: “Java”, “C++”, and “C” to account for the top three most frequently used languages and aggregated all the other languages into an “other” dummy variable.

**License Type:** OSS product license type may affect developer motivation and project success [32]. We control for license types to take into account potential project growth rate differences due to commercial or non-commercial nature of a project. License type is measured as a binary variable. All projects with GPL (general public license, usually indicating a non-commercial OSS product) license are given a value of 1. All other projects have a value of 0.

To examine the association between collaboration network structure metrics and project growth rate while taking into account project characteristics and control variables, we developed the regression model shown below as Equation 1. Here, all the control variables are represented by the general representation “controls”: The equation accounts for direct effect of the four metrics on the dependent variable, as well as interaction effect of the four metrics with the complexity of the development project as measured by the proxy for the construct software structural interdependence.

\[
\text{Project growth rate} = \beta_0 + \beta_1 \times \text{network-size} + \beta_2 \times \text{network-centralization} + \beta_3 \times \text{network-density} + \beta_4 \times \text{network-clusterness} + \beta_5 \times \text{software structural interdependence} + \beta_6 \times \text{software structural interdependence} \times \text{network-size} + \beta_7 \times \text{software structural interdependence} \times \text{network-centralization} + \beta_8 \times \text{software structural interdependence} \times \text{network-density} + \beta_9 \times \text{software structural interdependence} \times \text{network-clusterness} + \beta_1 \times \text{controls} + \varepsilon
\]

**Equation 1: Regression Equation**

In the model above, the \( \beta \) values represent the regression coefficients that show that impact of individual factors or their interaction with software structural interdependency on project growth rate. The model was tested against usual OLS assumptions and constrains like heteroskedasticity and multicollinearity, and found to not suffer from these problems.

As our model includes interaction terms as well as main effects with the interaction variable transformed to a binary scale, the interpretation of the coefficients needs elaboration. We can explain the appropriate interpretation using a simplified regression equation shown below:

\[
y = \beta_0 + \beta_1 \times x + \beta_2 \times z + \beta_3 \times x \times z
\]

**Equation 2: Simplified Regression Equation**

Here \( y \) represents the dependent variable – project growth rate, \( x \) represents the network structure measures and \( z \) represents software structural interdependence on a binary scale. We can see that for the case when structural interdependency is low (\( z = 0 \)), the marginal effect of \( x \) on \( y \) can be written as \( \frac{dy}{dx} = \beta_1 \). Now, when structural interdependency is high (\( z = 1 \)), the marginal effect of \( x \) on \( y \) can be written as \( \frac{dy}{dx} = \beta_1 + \beta_3 \). Hence, the incremental effect of an increase in structural interdependency on the relationship between a network structure measure \( (x) \) and project growth rate \( (y) \) is shown by the coefficient \( \beta_3 \) as \( \frac{\partial y}{\partial x} \times z = \beta_3 \). The significance of the main effect \( \beta_1 \) can be used to check for the effect of network structure metrics on project growth rate when structural interdependency is low. The significance of the interaction effect \( \beta_3 \) can be used to check the impact of increase in structural interdependency on relationship between network structure metrics and the project growth rate.

We first tested the regression model using ordinary least squares (OLS). We tested for standard assumptions of OLS estimation to ensure that our estimates are unbiased. As the model showed significant heteroskedasticity problem, we conducted robust regression and calculated Huber-White robust estimates of standard errors [34]. Finally, we conducted a likelihood ratio test to compare our research model (Equation 1) with a model of only main effects and no interaction variables. The null hypotheses of no difference between the two models was rejected (p-value < 0.001), showing that the
interactions add significant explanatory power to the model [35].

Preliminary Results

Our results are summarized below in the figure that lists effect of network measures on project growth rates for different structural interdependency levels and the marginal effect of increase in structural interdependency on the relationship between network measure and the project growth rate. The notations used in the table correspond to the Equation 1 presented before. All effects other than those indicated as “Not Significant” are statistically significant with p-value < 0.05. The figure shows the main effects ($\beta_1$ in Equation 1), interaction effects ($\beta_2$ ) and the net effect including both the main and the interaction effect ($\beta_1 + \beta_2$) for all four factors considered.

When structural interdependency is low, we only have the main effect of network clusterness represented by a negative and significant coefficient. This is represented by the solid downward sloping line. With increase in structural interdependency, the negative effect of network clusterness on project growth rate is mitigated since the interaction effect has a positive coefficient. This appears in the figure above as the positive change in the slope of the line as depicted by the arrow. Thus, we see that as structural interdependency increases, the negative effect of network clusterness on project growth rate indeed comes down. From managing the OSS project point of view, a project that involves developing software with low complexity, a collaboration structure with low levels of clusterness is more appropriate. In case the project has significant complexity, then the collaboration structure can allowed to be more clustered without sacrificing development performance.

5. Research and Practice Implications

Software engineering researchers have long focused on the organizational structure of individuals working on development tasks [7]. Role of such collaboration structure has been previously studied in traditional
software development [36] as well as OSS development projects [6]. In this paper we have emphasized the suitability of Social Network Analysis to analyze developer collaboration networks and leverage the resulting metrics to better manage the OSS development process. We explain the four key metrics as a starting point and present a proof of concept that shows the kind of inferences that can be drawn using the metrics.

This research has significant implications for practice as well. We described how the OSS development project leaders can generate the collaboration network metrics and use the metrics to better manage their projects. Our proof of concept analysis shows that for a given set of project characteristics and control variables, the metrics discussed can have significant impact on project growth rates. OSS project leaders can direct the development effort towards the appropriate direction (e.g. low clusterness for simpler development projects) to achieve higher project growth rates.

6. Conclusion

In this paper we have introduced an approach to monitor and manage OSS development efforts. The social network analysis method can be used to capture the unique self-organized nature of OSS development. We have shown how the social network analysis approach can be applied to an open source context and how collaboration networks and different network metrics can be generated. We then illustrate how these metrics capture different aspects of the OSS development process and what insights they may provide that can be used by the project leaders to effectively manage the project. We then presented a preliminary proof of concept analysis to show that the four metrics discussed have a significant impact on the success of OSS development process.

The research does suffer from several limitations. The research is based on data collected for a set of mature projects that may not be compatible with the current profile of OSS development projects. Further, the use of the analysis technique presented in this paper is limited by the ability of the OSS development team leadership to build the collaboration network and calculated the necessary network metrics. In this research we have not controlled for important aspects such as geographic dispersion of the developer community that may impact the network measures. Finally, it can be argued that OSS development is an organic process and even though a different collaboration structure may result in faster development, such a structure cannot be effectively imposed upon the developer community; limiting the practice impact of this research.

The use of social network analysis in open source software development is a growing research area and we expect that subsequent research will address the limitations of this research and further develop the approach.

7. References


