15 TOWARD A USER DRIVEN INNOVATION FOR DISTRIBUTED SOFTWARE TEAMS

Liaquat Hossain
David Zhou
The University of Sydney
Sydney, Australia

Abstract *The software industry has emerged to include some of the most revolutionized distributed work groups; however, not all such groups achieve their set goals and some even fail miserably. The distributed nature of open source software project teams provides an intriguing context for the study of distributed coordination. OSS team structures have traditionally been geographically dispersed and, therefore, the coordination of post-release activities such as testing are made difficult due to the fact that the only means of communication is via electronic forms, such as e-mail or message boards and forums. Nevertheless, large scale, complex, and innovative software packages have been the fruits of labor for some OSS teams set in such coordination-unfriendly environments, while others end in flames. Why are some distributed work groups more effective than others? In our current communication-enriched environment, best practices for coordination are adopted by all software projects yet some still fall by the wayside. Does the team structure have bearing on the success of the project? How does the communication between the team and external parties affect the project's ultimate success or failure? In this study, we seek to answer these questions by applying existing theories from social networks and their analytical methods in the coordination of defect management activities found in OSS projects. We propose the social network based theoretical model for exploring distributed coordination structures and apply that for the case of the OSS defect management process for exploring the structural properties, which induce the greatest coordination performance. The outcome suggests that there is correlation between certain network measures such as density, centrality, and betweenness and coordination performance measures of defect management systems such as quality and timeliness.*

Please use the following format when citing this chapter:

Hossain, L., and Zhou, D., 2008, in IFIP International Federation for Information Processing, Volume 287, Open IT-Based Innovation: Moving Towards Cooperative IT Transfer and Knowledge Diffusion, eds. León, G., Bernardos, A., Casar, J., Kautz, K., and DeGross, J. (Boston: Springer), pp. 261-270.

Keywords Open innovation, distributed teams, coordination, defect management process, software

1 INTRODUCTION

The open source approach to software development involves a group of loosely knit volunteers that collaborate over a public medium of communication, most popularly the Internet, to create software (de Souza et al. 2005). The source code is open to public access and it is this readily available nature that advocates of open source software claim results in faster and more responsive development cycles, thus producing more robust and secure software. These inherent characteristic have some interesting connotations for OSS project teams and how the software is tested. What is phenomenal about OSS projects is that their participants tend to form a community that is "bounded by their shared interest in using/developing the system" (Ye and Kishida 2003, p. 422).

In this study, the particular coordination activities in which we are interested are related to bug reporting, fixing, and knowledge sharing post-release of the OSS. Existing literature has looked into coordination of OSS projects; however, the general focus has been coordination between developers during the pre-release phase of the project. Minimal attempts have been make to include the OSS project community's involvement. The evolution of any OSS project originates from the input or identification of defects from the OSS project's community, therefore, it seems fitting to include them in the analysis of the network structure. Due to the distributed nature of OSS project teams and communities, the participants often are geographically scattered at arbitrary locations where the only means of coordinating tasks is via message boards, open forums, or e-mail. We explore questions such as: Does the degree of centrality, betweenness, and density of the network have any bearing on the number of defects fixed per software promotion? Does the degree of centrality, betweenness, and density of the network have any bearing on the number of defects reported at different severity levels? Does the degree of centrality, betweenness, and density of the network have any bearing on the average number of days for a defect to be fixed for each project team? Does the degree of centrality, betweenness, and density of the network have any bearing on the average number of days for responses by developers on defects?

2 DEFECT MANAGEMENT PROCESS AS A COORDINATED SYSTEM

Distributed software test teams are a recent trend that has emerged due to quality and ease of communication across vast physical locations and the boundless efforts of all corporations to reduce costs associated with software development. As a result, entire projects are now divided so that teams in charge of different phases of the project can be located oceans apart. The relevance of coordination theory comes to the forefront when we analyze distributed software testing because, apart from the multiple levels of communication from the tester to steering committee that must be coordinated, the actors

must contend with the physical distance between each team. By the using dependencies of coordination theory, we can deduce theoretical models that would help achieve the goal of software testing more effectively.

In order to provide some context on the coordinated system to which we are applying measures of coordination, it is only appropriate to outline the defect management process that distributed OSS teams must go through to ensure only legitimate and valid defects are accepted to be fixed. The defect management process is considered a coordinated system because it involves actors (developers and authors of the defect) performing interdependent tasks (raising and fixing defects) to achieve a common goal (advancing the OSS).

How do we know what is the most effective application of each coordination model? To answer this question, we need to apply our understanding of coordination components universal to all coordinated systems and whether the different methods of applying these components affect the results of existing metrics used to measure the effectiveness of software integration testing. Before we elaborate on the measures of OSS testing and how these relate to coordination, we need to step back and justify the validity of such metrics with regard to measuring coordination. We have previously established that OSS testing can be categorized as a coordinated system. This system comprises actors that perform interdependent tasks to achieve goals, the ultimate goal being to produce software to the community in a timely fashion, error free, and satisfying the requirements specified. In order to achieve these goals, the actors, working together and liaising with community members, must manage the components of coordination such as resource allocation, producer/consumer relationships, simultaneity constraints, and task dependencies. The measures of effectiveness include quality and timeliness, all of which are aligned with the ultimate goal of testing OSS projects.

Quality measures have a direct correlation with the producer/consumer relationship component of coordination theory. The core development team (producer) releases the software to the OSS community (consumer), then active or passive users log defect reports when they arrive at an incorrect or unexpected behavior from the system. The cause of the defect invariably relates to the producer/consumer relationship, particularly the prerequisite constraints, transfer, and usability aspects. To ensure that the software produced has a low quality of code metrics value and defect density metric value we must coordinate activities so that prerequisite constraints such as environment connectivity are correct. We must also ensure vital knowledge of the system is transferred from the development team to the test teams. The last element of usability will naturally fall into place once the previous elements are managed accurately.

The quality of code metric captures the relation between the number of weighted defects and the size of the product release.

$$\frac{W_F}{NCSPT}$$

Where W_F is number of weighted defect found post release software release. The weight of each defect is dependent on the severity. *NCSPT* is the number of *commit statements per time period.* In this metric, the lower the number is, the fewer defects or less serious defects found, thus the higher the quality of the code. Traditionally quality of code is measured using per thousand lines of code; however, this method of measure created

efficiencies such as "bloaty" (Mockus et al. 2002). With proprietary software, the common practice to minimize the effects of bloaty code was to only counting every thousand lines of new code. Also, the incremental delivery model of software development means that, with each release, only sections of new code are built on existing baseline code from the previous release. By only taking the new lines of code into consideration, a more accurate depiction of the quality of the code emerges. In the spirit of avoiding bloaty code to affect the accuracy of the measure of quality, this paper used NCSPT because OSS projects are tracked according to commit statements, which are logged and tracked on the OSS project website. Each commit statement indicates patches and fixes to the software, thus the proportionality between number of total defects raised and number of commit statement indicates if the quality of the code is high or low. Naturally, a high number indicates that many defects have arisen for each commit statement, which is a clear sign of poor quality of code.

The defect density metric shows the relation between numbers of weighted defects per severity level and the total number of defects detected post-release.

$$\frac{W_{PSL}}{W_F} * 100$$

Where W_{PSL} is number of weighted defects per severity level found during the post-release period of the OSS and W_F represents the total number of weighted defects found. Using this metric, we find that the higher the number, the higher the ratio of defects detected before release and the higher the effectiveness of the test. The motivating factor behind this measure is that we can arrive at a comparison based on number of defects of a certain severity level as a proportion of overall defect numbers. Naturally, projects with high quality will have fewer defects within the high range of defect severity. The successful result of the following timeliness metric can be associated the management of simultaneity constraints and task dependencies. Imagine a typical software testing phase, with absolute certainty there are dependencies on which activities can be performed first, in parallel, and last. It is up to those in the position to coordinate the activities to ensure that those tasks categorized as high priority because many other tasks depend on their success are run first. Those that can be run in parallel are organized in such a way, and those with many dependencies on other activities are run last.

Although coordination theory is a very powerful method of analyzing areas of improvement for OSS testing, there still exist limitations and questions that must be answered. For example, what effect does personal characteristics or properties of actors have on the outcome of the testing? Would group structure affect which activities can be performed? In a distributed network of groups that must perform tasks that are interdependent, what type of links induce the best overall outcome for the software integration testing phases? A social network based approach will help complete this picture and offer a more rounded model when combined with our coordination model for improving the outcomes of software integration testing. Density, as described earlier, is a measure of the number of connections formed within a network as a proportion of total possible connections between all nodes of a network. Studies into density have been conducted; Crowston and Howison (2006) found a negative correlation between the project size and density of the network. From their empirical work, it would seem that the connections for certain key participants remain the same while the project grows in size, thus making

the network less dense. From a coordination perspective, research has been done by Dinh-Trong and Bieman (2004) into comparing the defect density and general performance measures between different projects and seeing if indeed OSS produces better quality software. Therefore, it would be of interest to find the correlation, if any, between the network variable density and key performance indicators of OSS projects. In this paper, we test the following two assumptions:

H1: Quality of code measure is influenced by degree centrality, node betweenness, and density of the OSS team.

H2: Defect density measure is influenced by degree centrality, node betweenness, and density of the OSS team (high–medium–low severity)

3 DATA SOURCE

Data was collected via the monthly data-dump SF.net provided to the Department of Computer Science and Engineering, University of Notre Dame, for the sole purpose of supporting academic and scholarly research on the free/open source software phenomenon. We were granted access to the university's specialist wiki-supported website dedicated to the study of free/libre/open source software projects. This site provided the SF.net entity relationship diagrams, database schema definition, and SQL query functionalities required to extract the necessary data specified in the previous section. The SQL query tool was a web-based program that allowed for simple SQL queries and timed out when more complex queries were required to be executed. Therefore, the list of projects used for analysis was chosen from the 100 projects with the most active participation according to monthly download rate, forum activity, and pages view. From this list of 100 projects, only thpse with more than 7 developers and 200 bug reports were included. We felt that 7 developers provided a robust enough network for our studies and 200 bug reports represented enough instances of group interaction between the actors of the network. According to these criteria of selection, a group of 45 projects were identified to be appropriate for use in our analysis of the social network of an OSS team.

After the short list of projects to be included in the analysis was established, we went about extracting the necessary data from appropriate tables and fields according the ER-diagram and schema definitions provided by the University of Notre Dame. SQL queries were executed to extract the necessary data and the results of each query were saved as a text file with colons used as separators for each field. Each text file contains all closed status bug reports for one of the 45 projects that fit the selection criteria. The type of refinement needed for coordination measures of timeliness included conversion of the time-stamped fields such as artifact.open_date, artifact.close_date, and artifact_message. adddate, because SF.net uses UNIX formatted time-stamps for date fields. Unix time describes time as the number of seconds elapsed since midnight UTC of January 1, 1970, therefore, the following formula was applied in Excel to convert the time-stamp into a more meaningful manner.

$$\frac{UNIXtime + (365*70+19)*86400}{86400} - 0.41667$$

Once the time format had been changed, we used existing Excel functions to calculate the duration for each fix was closed according the artifact.open_date and artifact.close_date. Summing the total of the durations and dividing by the total number of bugs, we then have a coordination measure of time-to-fix-bugs metric for each project. A similar approach was used to calculate the time-to-respond metric, but the artifact_message.adddate was subtracted from the artifact.open_date with the average time similarly calculated by totaling the duration of each response and dividing by the total number of artifact_message.

For the measures of quality, defect density required aggregation of the different categories of the aftifact.priority field. Currently, SF.net's tracker system uses a 9-level priority system to categorize the severity of each bug, with 9 being the most severe and 1 being the least. This system presents ambiguity, when used in our analysis of the defect density, when defining the difference between a priority 5 bug and priority 4 or 6 bug. Thus to clarify the severity of the bugs raised for each project, we have concluded that due to the number of bugs found in each of the nine priority levels, one to three priority bugs are deemed "low severity," four to six priority bugs are referred to as "moderate severity," and seven to nine priority bugs are classified as "high priority." For the final measure of quality of code, total number of bugs raised and commit statements executed for each project were extracted from the project summary page.

4 RESULTS AND DISCUSSION

Using the multiple regression model of analysis, we arrived at a set of results that we can use to interpret the relationship between dependent variables of coordination and independent variables of social network characteristics. This section reports on the findings from this analysis. We analyze the regression output of the social network independent variables (centrality, betweenness, and density) regressed on both measures of quality of coordination respectively. From our analysis of the relationship between the quality of code measure and degree centrality, betweenness, and density, Excel processed the following regression model and output (Table 1):

$$Y = 2.593+11.361*\text{Centrality}+9.136*\text{Betweenness}-3.862*\text{Density}$$

We can only accept the hypothesis if for an F-test we obtain an F-statistic greater than 2.61 (critical value for F at 3, with 41 degrees of freedom) and, in addition, a significance-F value that is less than 0.05, which is the predetermined level of significance. In terms of the significance of the independent variables, each t-statistic needs be greater than 1.6829 or less than -1.6829 (critical value for a two-tailed t-test at 41 degrees of freedom) with p-values less than 0.05. If these conditions are met, then we can conclude that there is sufficient evidence of a relationship between quality of code and centrality, betweenness, and density. Interpreting the output, we accepted the hypothesis because the F value of 16.428 is greater than the critical value of 2.61 for a

Table 1. Quality of Code Output

Regression Statistics	
Multiple R	0.788236
R Square	0.62131599
A R Square	0.59909888
Standard Error	13.5134966
Observation	45

	Df	*SS*	*MS*	*F*	*Significance F*
Regression	3	8999.8732	2999.958	16.42781	0.0062626
Residual	41	7487.19761	182.6146		
Total	44	7735.61699			

	Coefficients	*Standard Error*	*t Statistic*	*P-value*
Centrality	11.3614902	6.25387107	1.816713	9.36E-16
Betweenness	15.1359258	6.39409508	2.367172	2.323E-12
Density	0.86176736	0.35632672	2.41848	0.000032

5 percent level of significance. Furthermore, we concluded that each independent variable has strong significance to the model with respective p-values all being greater than 1.6829, which is the critical value for a t-distribution test at a 5 percent level of significance. Therefore, each of the β_i's contribute to the model**.** The moderate adjusted R-squared suggests that the model as a whole has average explanatory power and around 60 percent of the variations in observed values for quality of code can be explained by the variations of values of centrality, betweenness, and density. From the strong p-values of the independent variables, we can conclude the following:

1. For each percentage increase of centrality, on average it will increase the quality of code measure by 11.361 units, holding all other variables constant.
2. For each percentage increase of betweenness, on average it will increase the quality of code measure by 15.136 units, holding all other variables constant.
3. For each percentage increase of density, on average it will decrease the quality of code measure by 3.862 units, holding all other variables constant.

What these conclusions illustrate is that while centrality and betweenness of the network have a positive correlation with quality performance of OSS teams, interestingly, density has a negative correlation. This result suggests that increasing the level of centrality and betweenness of an OSS team will increase the number of bugs fixed per commit statement. However, increasing the communication links between actors will actually decrease this measure of quality in the coordination of defect management activities. For the measure of defect density, three separate regression analyses were conducted according to the three levels of defect that were classified for each OSS project. This is to take into the account of the mitigating factor of defect severity levels. As also previously discussed, when a defect is recognized as valid, it is assigned a severity or priority level, which is an indication of the effect this defect has on the project. Naturally, higher severity defects demand more attention because they affect the behavior

of the software to a greater extent. For this reason, we decided to conduct separate regression analyses on the three levels of defect severity to see what relationships exist between that and centrality, betweenness, and density. The regression analysis produced the following model and output (Table 2):

$$Y = 2.593+11.361*\text{Centrality}+9.136*\text{Betweenness}-3.862*\text{Density}$$

Here, we can only accept the hypothesis if for an F-test we obtain an F-statistic greater than 2.61 (critical value for F at 3, with 41 degrees of freedom), and, in addition, the significance-F value is less than 0.05, which is the predetermined level of significance. In terms of the significances of the independent variables, each t-statistic needs be greater than 1.6829 or less than -1.6829 (critical value for a two-tailed t-test at 41 degrees of freedom) and p-values less than 0.05. If these conditions are met, then we can conclude that there is sufficient evidence of a relationship between quality of code and centrality, betweenness, and density. Interpreting the output, we can only tentatively accept H2(a) because the F value of 4.310 is greater than the critical value of 2.61 for a 5 percent level of significance. However, each independent variable shows poor insignificance to the model with their respective p-values all being greater than -1.6829 and less than 1.6829, which are the critical values for t-distribution test at 5 percent level of significance. Therefore, none of the β_i's contribute to the model. Furthermore, the low adjusted R-squared suggests that the model as a whole has poor explanatory power and only around 30 percent of the variations in observed values for defect density for high severity defects can be explained by the variations of values of centrality, betweenness, and density.

These results suggest that we need to do further analysis into the model and the decision was taken to include an interaction term into the model. The adjusted model included the introduction of centrality*density and betweenness*density. These two inter-

Table 2. High Severity Defect Output

Regression Statistics					
Multiple R	0.4828337				
R Square	0.2331284				
A R Square	0.1998723				
Standard Error	0.0008997				
Observation	45				
	Df	*SS*	*MS*	*F*	*Significance F*
Regression	3	0.22459875	0.07486624	3.82518	0.01665228
Residual	41	0.80244947	0.01957193		
Total	44	1.02704823			
	Coefficients	*Standard Error*	*t Statistic*	*P-value*	
Centrality	-0.077590007	0.08544898	-0.9080273	0.176248	
Betweenness	-0.096314342	0.08690062	-1.1083269	0.322982	
Density	-0.008513927	0.0758123	-0.1123030	0.017061	

action terms were chosen because networks with high degrees of density tend to have lower degrees of centrality and betweenness, due to the increase in connections between actors, thus reducing the centralization measure of individual actors. Intuitively this assumption can be made because as the ties among actors increase there is less need for actors to communicate through other actors; instead, they are more likely to contact the source of the knowledge, consequently reducing the need for central or between actors.

5 CONCLUSIONS

This study suggests that there is a correlation between social network characteristics and strong and poor performing projects in an OSS environment. The projects that were analyzed using our theoretical model all fit the criteria of more than 7 developers and more than 200 defect reports. The conditions we set on the data have helped to validate the strength of the results because such criteria provide a robust network of interaction between actors and facilitate varying sizes of networks. Analysis of the data displayed a normal distribution, which fit our proposed parameterized regression analysis. We believe that we have made a significant contribution into the study of distributed work groups, social network analysis, and coordination. Studies by Madey et al. (2002) have investigated the social network phenomenon, the power-law relationship in OSS development teams, and suggested looking into degrees of separation of the connection which we have analyzed through density and centrality.

References

Crowston, K., and Howison, J. 2006. "Hierarchy and Centralization in Free and Open Source Software Team Communication," *Knowledge, Techology and Policy* (81:4), pp. 65-85.

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*, Sanibel Island, FL, November 6-9, pp. 197-206.

Dinh-Trong, T., and Bieman, J. M. 2004. "Open Source Software Development: A Case Study of FreeBSD," in *10th IEEE International Symposium on Software Metrics*, Los Alamitos, CA: IEEE Computer Society, pp. 96-104.

Madey, G., Freeh, V., and Tynan, R. 2002. "The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory," in *Proceedings of the Eighth Americas Conference on Information Systems*, R. Ramsower and J. Windsor (eds.), Dallas, TX, August 8-11, pp. 1806-1813.

Mockus, A., Fielding, R. T., and Herbsleb, J. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology* (11:3), pp. 309-346.

Ye, Y., and Kishida, K. 2003. "Towards an Understanding of the Motivation Open Source Software Developers," in *Proceedings of the 25th International Conference on Software Engineering*, Portland, OR, May 3-10, pp. 419-429.

About the Authors

Liaquat Hossain's work aims to explore the effects of different types of social network structures and patterns of information technology use on coordination in a dynamic and complex environment. The primary focus of Liaquat's research is in the area of network analysis of organizational and social systems. He approaches this using social networks theory and analytical methods and applies theories and methods from sociology and social anthropology to study coordination problems in a dynamic, complex, and distributed environment. He further applies network-based theories and methods to explore the phenomenon of globally distributed work groups (referred to as outsourcing in business management literature) and its management challenges. Overall, he is interested in exploring (modeling and empirical investigation) the effects of different types of social network structures on coordination and organizational performance from a theoretical and applied perspective. In his research, he uses methods and analytical techniques from mathematical sociology (i.e., social networks analysis), social anthropology (i.e., interview and field studies), and computer science (i.e., information visualization, graph theoretic approaches, and data mining techniques such as clustering) to explore coordination problems in a dynamic, distributed and complex setting. Liaquat can be reached by e-mail at lhossain@it.usyd.edu.au.

Davis Zhao received his Bachelor of Engineering in Electrical and Information Engineering specializing in Software Engineering from the University of Sydney in 2008. He is currently working as a consultant at Accenture Australia.