On The Study of Open Source Software to Inform the Design of CSCW in Global Software Development

Michelle W. Purcell
Drexel University
3141 Chestnut St.
Philadelphia PA 19104
mjw23@drexel.edu

ABSTRACT
Global software development (GSD) is struggling with issues related to globally distributed work, such as coordination, maintaining awareness, and knowledge sharing. Open Source Software, however, appears to have overcome these issues and can be a source of knowledge for advancing GSD. This position paper examines existing CSCW literature related to collaborative practice in OSS to identify areas for further investigation that have potential to improve the conduct of invisible work in GSD.

Author Keywords
Global Software Development; Open Source Software; Collaboration

ACM Classification Keywords
H.5.3. Group and Organization Interfaces: Computer-Supported Cooperative Work

INTRODUCTION

Global software development (GSD) is struggling with issues related to the globally distributed work including issues of coordination, maintaining awareness, and knowledge sharing. Underlying such problems is the move from co-located to geographically distributed work, which significantly impacts collaboration [8]. Issues of group coordination, group awareness, and mechanisms to underpin collaboration with shared knowledge are not addressed well in current practice.

While GSD still struggles with these issues, open source software (OSS), which is geographically-distributed work often including developers from around the world, appears to have overcome them [6,13,15]. There are many success stories from Apache and Firefox to lesser known but no less important projects producing humanitarian free and open source (HFOSS) software, such as OpenMRS, a medical records system for developing countries. To understand the extent of OSS use, the Internet infrastructure mostly runs on open source products. For instance, Apache Tomcat is used on over 60 percent of websites [9]. With regard to its robustness, OSS is traditionally better than proprietary systems in terms of portability and given the openness of code it also generally more reliable and secure; a famous saying related to open source, "Given enough eyeballs, all bugs are shallow," [12] attests to that. There is some evidence to back these claims; for example, an empirical study of MySQL showed six times less defects than similar commercial databases [14].

As such, given the traceability of OSS practice – much of which is open for study from artifacts available on the Internet through project use of tools such as mailing lists, Internet Relay Chat (IRC), web sites, and code repositories – OSS provides an opportunity to learn how to overcome issues of collaboration across distance. Furthermore, companies have interest in utilizing OSS practices within their own distributed teams, but are uncertain on how to apply in their own contexts [1,5]. This paper first presents CSCW research on OSS practice and then provides recommendations for research on the study of OSS to aid with the conduct of invisible work in GSD.

ON THE STUDY OF INVISIBLE WORK IN OSS

CSCW literature has investigated OSS practice across multiple contexts, which can be organized in three areas: communication tools; methods of promoting awareness; and organizational aspects of OSS. Strewn throughout are examples of how invisible work is handled in OSS.

Impacts of Media or Platform Used For Project Communication

Yamauchi et al. [15] examined development and communication practice among two open source projects, FreeBSD Newconfig and GNU GCC. Projects were chosen to explore practice for two different types of tasks, new development and maintenance. From a technical perspective the paper presents different tools used and their roles as knowledge sharing mechanisms. The use of a source code management tool served as a boundary object

for providing shared context but enabling work to remain loosely coupled across individuals. To-do lists served as maps and scripts, enabling participants to build effective mental models of collaboration needs**.** The practice of broadcasting information on mailing lists was used to maintain awareness**.**

Cultural practices were identified from the analysis of communication practices by examining the content and transition between messages. Because of the lean aspects of the communication tools—for instance, a reduced social context when using email or mailing lists—asynchronous tools in use oriented collaborative design decisions towards rationality by allowing developers time to reflect and craft arguments. Developers justified their decisions by making behavior "logically plausible" [15]. But lean communication tools also made it difficult to plan before action, e.g., writing a patch, performing a test. This created a bias toward action, with discussion occurring following the results, e.g., test results provided fodder for more fruitful dialogue.

Despite examples of developers' ability to adjust communication practice to overcome limitations of the tools, Brue et al. [2] found that was not possible during communications between developers and users during bug fix, "developers get annoyed and impatient over incomplete bug reports and users are frustrated when their bugs are not immediately fixed." Through quantitative and qualitative analysis of questions asked and answered within 600 bug reports from Eclipse and Mozilla the authors sought to understand information needs as well as the relationship between aspects of the bug report and response rate, response time and question time. Problems arose due to lack of user involvement and misunderstanding of developer needs during bug fix. In addition, in cases of low response rate they found examples where it was due to questions of triaging, debugging and reproducing bugs. With regard to improving user response it was recommended to draw users' attention to what is specifically needed from them and motivate participation through making using the system more of a social activity.

These examples show that in some tasks projects are able to overcome difficulties with distributed work by boundary object use that reduces dependencies between individuals. However, in cases where that is not possible, communicative practices needed to be modified to overcome the limitations of the tools used, which must be promoted community culture. This can be seen by suggestions by Breu et al. [2] to make the bug reporting tool used more like a social activity as perhaps users' involvement is not motivated by the same cultural practice as the project to which they are contributing bug reports.

Collaborative Practices To Maintain Group Awareness

Gutwin et al. [6] examined how group awareness was maintained among the NetBSD, Apache httpd, and Subversion open source projects. His findings suggest that mechanisms outside code partitioning to loosely couple work were at play in overcoming collaborative difficulties associated with distributed group work. Projects used IRC, mailing lists, and commit logs to help developers know who the experts were, who was working on what and to support informal and formal communication. As important, however, was the finding that organizational culture must promote behavior that makes group awareness possible using these tools. While generally effective the authors raised questions of transporting the tools to other distributed work groups. Because OSS developers are usually "cream of the crop" it may be they are more capable using and monitoring multiple tools to maintain awareness.

While most OSS projects use hosting services for source code repositories, relatively recently there has been a movement to GitHub, a site that provides source code management through git, additional trace data that provides awareness, and social networking capabilities. Dabbish et al. [3] inquire into the collaboration and learning benefits available for open source developers and projects using GitHub. Much of what was studied relates to facilitating peer production in terms of identifying viable projects, capable contributors and user needs; however, trace data also made it possible to identify experts and know who was working on what. However, despite the availability of trace data and added social context, communication was necessary in cases of a lack of transparency, "Thus although passive activity traces of others' behavior are powerful in some ways, they are limited when joint action is required. In part this is because of the lack of feedback or interactivity these visible traces provide." [3] These were situations when providing rationale, plans and mutual negotiation were necessary, such as managing cross-project dependencies and negotiating changes to contributions. In those cases users had to move to tools outside GitHub. Marlow et al. [11] also studied GitHub but for understanding how impressions are formed about contributors. Impressions influence project owners' receptivity to contributions, and as such they may be less inclined to accept contributions from persons perceived as newcomers or less skilled.

As Gutwin et al. [6] describes the use of different tools for different group awareness functions, we see GitHub being used to provide a different type of awareness mainly related to becoming aware of user needs and project dependencies; the use of reputation indicators for projects to assess developer contributions; and for users to identify viable projects to contribute to. In addition, we see further evidence of trace data alone not being sufficient to facilitate coordination as in the case of times when plans and rationale needed to be discussed. So, while GitHub was useful to maintain types of awareness that mailing lists, IRC, and code commits could not provide it was yet another tool that had to be monitored to maintain awareness. As suggested in [6] since OSS developers tend to be the

exceptional lot the addition of yet another tool to monitor raises questions of feasibility in situations of developers with lesser skill.

Organizational Aspects of OSS

Ducheneaut [4] and Sack et. al [13] approach the study of OSS seeking to understand both the social and technical aspects of practice together through the use of a specially designed methodological framework. Ducheneaut [4] describes new member enculturation into the community not as static roles but as a trajectory and presents the technical as well as social behavior contributors must undertake to move from peripheral to core contributors. Sack et al. [13] discuss looking across the three information spaces of OSS – implementation (e.g., code repository), documentation space, and discussion space (e.g., mailing lists) in various combinations to understand power dynamics in the design process, new member enculturation, and how to maintain thematic coherence during design discussions. These papers try to advance understanding of the role of organizational structure during communication in OSS. For instance, while Yamauchi et al. [15] describe a pattern of communication during the design process Sack et al. [14] extend it by presenting roles of participants as well as activities performed by them. By understanding roles and activities of participants in the Python community enhancement process, one can see how thematic cohesion is maintained during asynchronous design discussion.

DISCUSSION

There are three takeaways from the CSCW literature on OSS that present interesting research avenues with potential for improving the conduct of invisible work in GSD. First, coordination dependencies between developers are reduced due to modularization of code, tool use, and work practices. Second, when coordination is required there is a plethora of tools that need to be monitored to maintain awareness. Third, OSS communities enact communication practices to overcome limitations of the leanness of communication tools. This section discusses those points in further detail and presents areas for further research.

While Yamauchi et al. [15] found that the source code management tool worked as a boundary object that enabled loose coupling, Hemetsberger and Reinhardt [7] and Bolici et al. [1] in examining developer -to -developer interaction discovered that collaboration occurred through the use of a trading zone [10] where work was accomplished through display, representation, and assembly. Display involves making code and communication available to everyone. Representation entails the use of coding processes that enable correct interpretation of the code by members of the community. Lastly, assembly involves adding to existing code uploaded by other developers [1].

The trading zone approach works to enable stigmergic coordination where, "The action of an actor produces changes in the environment, and these changes can provide a stimulus for other actors, who respond with another action, triggered by the previous one, and so on. Thus the traces left by an individual, or the result of its work, can act as a direct source of stimuli for others." [1] It is known that coordination is affected by software architecture, however we know little about the relationship itself [8]. One can infer therefore that in OSS when stigmergic coordination is achieved the source code architecture should exhibit characteristics that support it as well as tool characteristics and work processes. Research on software architecture and coordination, tool affordances, and tool use practice in different OSS contexts could be used to inform GSD when looking to reduce issues of developer coordination across distance by reducing dependencies.

Gutwin et al. [6] discussed how OSS developers used multiple tools including IRC, mailing lists, and commit logs to maintain awareness, however, with the caveat that maintaining awareness may be time consuming and difficult for less experienced or capable developers. It was questioned as to whether such an approach can be transported to other virtual settings. Further research should investigate the information contained across the multiple tools from an assembly perspective from which to understand the awareness process and how it could be improved.

Lastly, Yamauchi et al. [15] and Sack et al. [13] both discuss patterns of communication that are employed in OSS to overcome limitations of asynchronous communication. Further research of OSS communication behaviors should be examined to help inform communicative practice in GSD. In addition, such research may point to opportunities for augmenting or redesigning existing asynchronous communication tools in support of distributed software development.

CONCLUSION

The study of OSS practice enables opportunities for industrial GSD to adopt similar practices to overcome issues of geographic distribution. While there has been some study on OSS practice in CSCW, it is not clear how lessons learned from OSS can be applied to GSD, given the socio-technical complexity of OSS systems. This paper presents some research suggestions to help bridge that gap. In particular, it discusses the need for further study of how coordination dependencies are reduced in OSS, ways to improve the maintenance of awareness across multiple software tools, and investigation of the applicability of OSS communicative practices to reduce issues with distributed collaboration.

REFERENCES

1. Bolici, F., Howison, J., & Crowston, K. (2009). Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects. In *Socio-Technical Congruence Workshop in Intl Conf on Software Engineering (2009).*

2. Breu, S., Premraj, R., Sillito, J., & Zimmermann, T. (2010). Information needs in bug reports: improving cooperation between developers and users. In *Proc. CSCW 2010, ACM Press (2010)*.
3. Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. CSCW 2012*.
4. Ducheneaut, N. (2005). Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW), 14*(4), 323–368.
5. Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2005). A case study of open source tools and practices in a commercial setting. In *ACM SIGSOFT Software Engineering Notes* (Vol. 30, pp. 1–6).
6. Gutwin, C., Penner, R., & Schneider, K. (2004). Group awareness in distributed software development. In *Proc. CSCW 2004, ACM Press (2004)*.
7. Hemetsberger, A., & Reinhardt, C. (2009). Collective development in open-source communities: An activity theoretical perspective on successful online collaboration. *Organization studies, 30*(9), 987–1008.
8. Herbsleb, J. D. (n.d.). Global Software Engineering: The Future of Socio-technical Coordination. In *Proc. Future of Software Engineering (FOSE)*.
9. Historical trends in the usage of web servers, September 2012. (n.d.). Retrieved September 4, 2012, from http://w3techs.com/technologies/history_overview/web_server
10. Kellogg, K. C., Orlikowski, W. J., & Yates, J. (2006). Life in the trading zone: Structuring coordination across boundaries in postbureaucratic organizations. *Organization Science, 17*(1), 22–44.
11. Marlow, J., Dabbish, L., & Herbsleb, J. (2013). Impression formation in online peer production: activity traces and personal profiles in github. In *Proc. CSCW 2013, ACM Press (2013)*.
12. Raymond, E. (n.d.). The Cathedral and the Bazaar. Retrieved December 13, 2012, from http://www.catb.org/esr/writings/homesteading/cathedral-bazaar/
13. Sack, W., Détienne, F., Ducheneaut, N., Burkhardt, J. M., Mahendran, D., & Barcellini, F. (2006). A methodological framework for socio-cognitive analyses of collaborative design of open source software. *Computer Supported Cooperative Work (CSCW), 15*(2), 229–250.
14. Tong, T. W. (2004). Free/open source software education. *United Nations Development Programme's Asia-Pacific Information Programme, Malaysia*.
15. Yamauchi, Y., Yokozawa, M., Shinohara, T., & Ishida, T. (2000). Collaboration with Lean Media: how open-source software succeeds. In *Proceedings of the 2000 ACM CSCW 2000, ACM Press (2000)*.

.