Effective Work Practices for Software Engineering: Free/Libre Open Source Software Development

Kevin Crowston
Syracuse University
School of Information Studies
Syracuse NY USA
+1 315 443–1676
crowston@syr.edu

Hala Annabi
The Information School
University of Washington
Seattle, WA, USA
+1 206 616-8553
hpannabi@u.washington.edu

James Howison & Chengetai Masango
Syracuse University
School of Information Studies
Syracuse NY USA
+1 315 443–4508
{jhowison,cmasango}@syr.edu

ABSTRACT

We review the literature on Free/Libre Open Source Software (FLOSS) development and on software development, distributed work and teams more generally to develop a theoretical model to explain the performance of FLOSS teams. The proposed model is based on Hackman's [34] model of effectiveness of work teams, with coordination theory [52] and collective mind [79] to extend Hackman's model by elaborating team practices relevant to effectiveness in software development. We propose a set of propositions to guide further research.

Categories and Subject Descriptors: D.2.9 **[Software Engineering]**: Management—*programming teams*

General Terms: Management

Keywords: Collective mind theory, Coordination theory, Free and open source software, Team effectiveness.

1. INTRODUCTION

This paper presents a research approach to studying software engineering as a team-centered work task. Our research employs interdisciplinary approaches drawing on information systems, distributed work and small groups research to examine effective work practices for software development teams, specifically those engaged in the development of Free/Libre and Open Source software (FLOSS). It is our belief that understanding the social and socio-technical practices of FLOSS development teams can provide insights that are useful for understanding software engineering as a human practice throughout the field.

Free/Libre Open Source Software (FLOSS) is a broad term used to embrace software developed and released under an "open source" license allowing inspection, modification and redistribution of the software's source.[1] There are thousands of FLOSS projects, spanning a wide range of applications. Due to their size, success and influence, the Linux operating system and the Apache Web Server are the most well known, but hundreds of others are in widespread use, including projects on Internet infrastructure (e.g., sendmail, bind), user applications (e.g., Mozilla, OpenOffice) and programming languages (e.g., Perl, Python, gcc). Many are popular (indeed, some dominate their market segment) and the code has been found to be generally of good quality [69].

Key to our approach is the fact that most FLOSS software is developed by self-organizing distributed teams. Developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily by means of computer-mediated communications (CMC) [64, 77]. These teams depend on processes that span traditional boundaries of place and ownership. The research literature on software development and on distributed work emphasizes the difficulties of distributed software development, but the case of FLOSS development presents an intriguing counter-example. What is perhaps most surprising about the FLOSS process is that it appears to eschew traditional project coordination mechanisms such as formal planning, system-level design, schedules, and defined development processes [38].

As well, FLOSS development is an important phenomena deserving of study for itself. FLOSS is an increasingly important commercial phenomenon involving all kinds of software development firms, large, small and startup. Millions of users depend on systems such as Linux and the Internet (heavily dependent on FLOSS tools), but as Scacchi [66] notes, "little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success". A 2002 EU/NSF workshop on priorities for FLOSS research identified the need both for learning "from open source modes of organization and production that could perhaps be applied to other areas" and for "a concerted effort on open source in itself, for itself" [27].

Certainly, FLOSS is a growing component of software engineering and the overlap between FLOSS development and traditional in-house and proprietary development is increasing. It is clear that firms are seeking to leverage FLOSS code to bolster their competitive positions but there are opportunities to learn from the social and socio-technical practices of FLOSS development teams to improve the effectiveness of software engineering as a human and team practice. We recognize that the FLOSS community comprises teams and practices with significant differences, and that these practices overlap with proprietary software engineering,

[1] The software is generally available without charge ("free as in beer"). Much (though not all) of it of is also "free software", meaning that derivative works must be made available under the same license terms ("free as in speech", thus "libre"). We have chosen to use the acronym FLOSS rather than the more common OSS to acknowledge this dual meaning.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WISER'04, November 5, 2004, Newport Beach, California, USA.
Copyright 2004 ACM 1-58113-988-8/04/0011...$5.00.

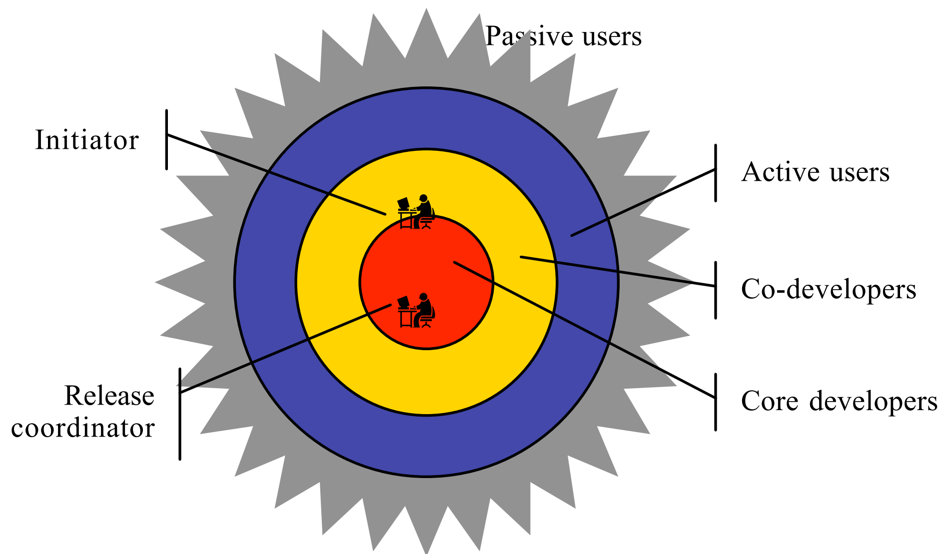

Figure 1. Hypothesized FLOSS development team structure.

especially when developers are pursuing agile software development.

2. CURRENT RESEARCH ON FLOSS

The nascent research literature on FLOSS has addressed a variety of questions. First, researchers have examined the implications of FLOSS from economic and policy perspectives. For example, some authors have examined the implications of free software for commercial software companies or the implications of intellectual property laws for FLOSS [e.g., 18, 43, 50]. Second, various explanations have been proposed for the decision by individuals to contribute to projects without pay [e.g., 3, 23, 36, 40, 53]. These authors have mentioned factors such as personal interest, ideological commitment, development of skills [51] or enhancement of reputation [53]. Finally, a few authors have investigated the processes of FLOSS development [e.g., 64, 70], which is the focus of this paper.

Raymond's [64] bazaar metaphor is the most well-known model of the FLOSS process. As with merchants in a bazaar, FLOSS developers are said to autonomously decide how and when to contribute to project development. By contrast, traditional software development is likened to the building of a cathedral, progressing slowly under the control of a master architect. While popular, the bazaar metaphor has been broadly criticized. According to its detractors, the bazaar metaphor disregards important aspects of the FLOSS process, such as the importance of project leader control, the existence of de-facto hierarchies, the danger of information overload and burnout, and the possibility of conflicts that cause a loss of interest in a project or forking [4, 5].

Recent empirical work has begun to illuminate the structure and function of FLOSS development teams. Gallivan [26] analyzes descriptions of the FLOSS process and suggests that teams rely on a variety of social control mechanisms rather than on trust. Several authors have described teams as having a hierarchical or onion-like structure [9, 24, 56], as shown in Figure 1. At the centre are the core developers, who contribute most of the code and oversee the design and evolution of the project. The core is usually small and exhibits a high level of interaction, which would be difficult to maintain if the core group were large. Surrounding the core are the co-developers. These individuals contribute sporadically by reviewing or modifying code or by contributing bug fixes. The co-developer group can be much larger than the core, because the required level of interaction is much lower. Surrounding the developers are the active users: a subset of users who use the latest releases and contribute bug reports or feature requests (but not code). Still further from the core are the passive users. The border of the outer circle is indistinct because the nature and variety of FLOSS distribution channels makes it difficult or impossible to know the exact size of the user population. As their involvement with a project changes, individuals may move from role to role. However, core developers must have a deep understanding of the software and the development processes, which poses a significant barrier to entry [22, 37]. This barrier is particularly troubling because of the reliance of FLOSS projects on volunteer submissions and "fresh blood" [15]. It is important to note that this description of a project team (Figure 1) is based on a few case studies. While the model has good face validity, it has not been extensively tested.

The other major stream of research examines factors for the success of FLOSS in general (though there have been few systematic comparison across multiple projects, e.g., [71]). The popularity of FLOSS has been attributed to the speed of development and the reliability, portability, and scalability of the resulting software as well as the low cost [12, 35, 49, 62, 63, 73, 74]. In turn, the quality of the software and speed of development have been attributed to two factors: that developers are also users of the software and the availability of source code.

First, FLOSS projects often originate from a personal need [55, 75], which attracts the attention of other users and inspire them to contribute to the project. Since developers are also users of the software, they understand the system requirements in a deep way, eliminating the ambiguity that often characterizes the traditional software development process: programmers know their own needs [46]. (Of course, over-reliance on this mode of requirements gathering may also limit the applicability of the FLOSS model.)

Second, in FLOSS projects, the source code is open to modification, enabling users to become co-developers by developing fixes or enhancements. As a result, FLOSS bugs can be fixed and features evolved quickly. Active users also play an important role [61]. Research suggests that more than 50 percent of the time and cost of non-FLOSS software projects is consumed by mundane work such as testing [68]. The FLOSS process enables hundreds of people to work on these

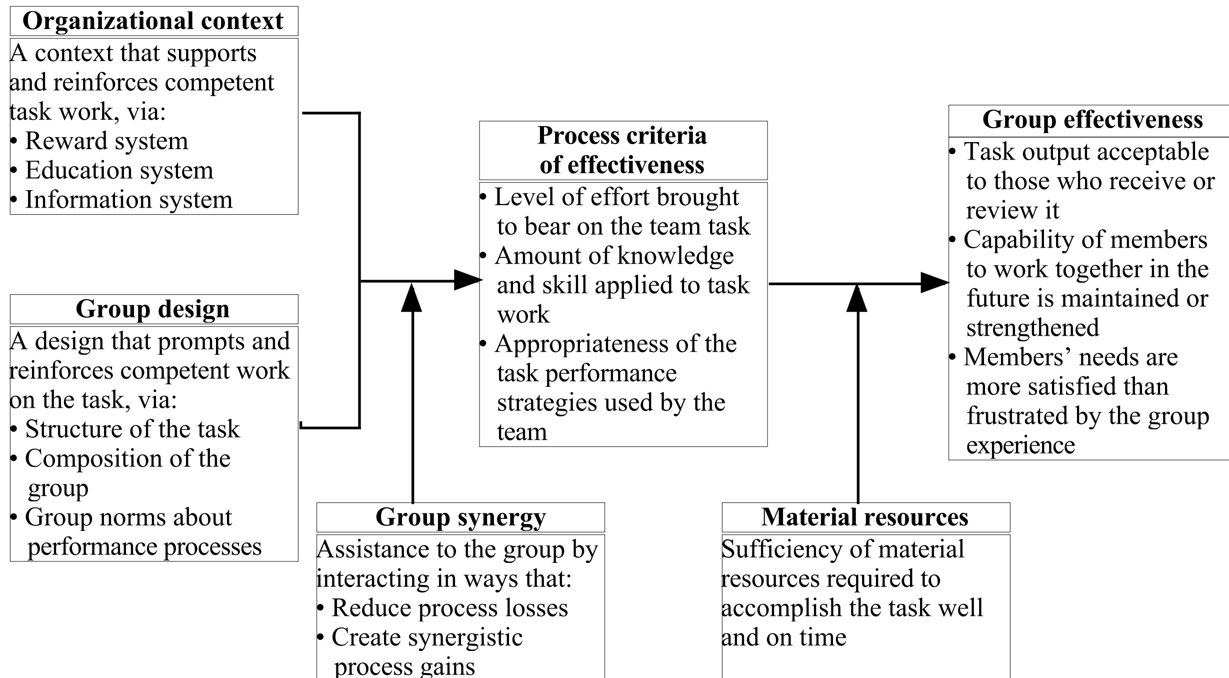

Figure 2. Hackman's [34] normative model of group effectiveness.

parts of the process [48]. Intriguingly, it has been argued that the distributed nature of FLOSS development may actually lead to more robust and maintainable code. Because developers cannot consult each other easily, it may be that they make fewer assumptions about how their code will be used and thus write more robust code that is highly modularized [48].

It is noteworthy that much of the literature on FLOSS has been written by developers and consultants directly involved in the FLOSS community. These contributions are significant as they point out the economic relevance of FLOSS as well as the most striking aspects of the new development process. Yet many of these studies seem to be animated by partisan spirit, hype or skepticism [29]. There are only a few well-documented case studies [e.g., 54], most of which discuss successes rather than failures. Finally, with a few exceptions [e.g., 1, 81], the proposed models are descriptive and based on a small number of cases. This is both indicative of the relative novelty of the issue and the lack of a clear theoretical framework to describe and interpret the FLOSS phenomenon [13]. Our work is intended to fill some of these gaps by providing a theoretically-based model of FLOSS development practices.

3. WORK TEAM EFFECTIVENESS FOR SOFTWARE DEVELOPMENT

We are interested in studying work practices that make FLOSS projects more effective. To do so, we have chosen to analyze developers as comprising a work team. Much of the literature on FLOSS has conceptualized developers as forming communities, which is a useful perspective for understanding why developers choose to join or remain in a project. However, for the purpose of this study, we view the projects as entities that have a goal of developing a product, whose members are interdependent in terms of tasks and roles, and who have a user base to satisfy, in addition to having to attract and maintain members. These aspects of FLOSS projects suggest analyzing them as work teams. Guzzo and Dickson [33, pg. 308] defined a work team as "made up of individuals who see themselves and who are seen by others as a social entity, who are interdependent because of the tasks they perform as members of a group, who are embedded in one or more larger social system (e.g. community, or organization), and who perform tasks that affect others (such as customers or coworkers)".

Given this perspective, we draw on Hackman's [34] model of effectiveness of work teams as a conceptual basis for our study. While this model was initially presented as sets of factors, these factors point to work practices that are important for team effectiveness. Following on Crowston and Kammerer [11], we use coordination theory [52] and collective mind [79] to extend Hackman's model by further elaborating team practices relevant to effectiveness in software development. In this section, we describe these theories, their applicability to FLOSS development and develop a set of propositions for future work.

3.1 Team effectiveness model

Researchers in social and organizational psychology have studied teams and their performance for decades and have developed a plethora of models describing and explaining team behavior and performance. One of the most widely used normative models was proposed by Hackman [34], shown in Figure 2. Hackman's [34] model is broadly similar to other models [30], such as [44], [28] or [59]. However, Hackman's model seems especially fitting because of its intended purpose of identifying factors related to team effectiveness, broadly defined, and its inclusion of team process factors.

3.1.1 Outputs.

Hackman's [34] model is presented in an input-process-output framework. The output explained by the model is team

effectiveness, which is clearly a key variable for our study: if we cannot distinguish more and less effective teams, we cannot identify work practices related to effectiveness. An attractive feature of Hackman's [34] model is that effectiveness is conceptualized along multiple dimensions, not just task output. Hackman also includes the team's continued capability to work together and satisfaction of individual team members' personal needs as relevant outputs. These three types of output correspond well to the effectiveness measures for FLOSS projects identified by Crowston, Annabi and Howison [10], who proposed measures including system quality (task output), developer satisfaction (satisfaction of individual needs), and number of developers, developer turnover and progress of the project through stages of development (e.g., alpha to beta to production), all indicative of the continued ability of the team to work together. These measures had been de-emphasised in the management-oriented Information Systems literature (eg. Delone).

> ***Definition:*** Effectiveness for FLOSS teams can be measured by creation of quality software, continued team performance and team member satisfaction.

3.1.2 Inputs.

Hackman's model includes two sets of input factors, *organizational context* (reward, educational and information systems) and *group design* (task structure, team composition and team norms).

The *organizational context* factors seem possibly important, though FLOSS teams typically mix members from a variety of organizational contexts, so these contextual factors may not be under the control of the projects. As a result, we do not include these factors in our theorizing.

Instead, we plan to focus initially on *team design*, which includes three promising factors to explore: task structure, team composition and team norms.

- All FLOSS teams perform much the same task, namely software development, but we anticipate seeing important differences in the way teams *structure the task*. To analyze these structures, we will use coordination theory (discussed below).
- Based on the review above, we anticipate seeing differences in practices related to team composition. In particular, prior research has suggested the importance of having contributions from members in different roles, such as core members, co-developers and active users.

> ***Proposition:*** Teams with members contributing in all roles will be more effective.

- Finally, we anticipate differences in the development of *team norms*, in particular, in the way new members are socialized into and contribute to teams (as discussed below).

3.1.3 Process.

The intermediary factors in Hackman's model are three process criteria (i.e., indications that the process is working as it should): "the *level of effort brought to bear on the team task*, *amount of knowledge and skill applied to task work*, and *appropriateness of the task performance strategies used by the group*" [34].

Prior work has noted that distributed teams often need to expend more on-task *effort* to be effective, suggesting the importance of this variable. More effort is required for interaction when participants are distant and unfamiliar with each others work (Ocker and Fjermestad 2000; Seaman and Basili 1997). The additional effort required for distributed work often translates into delays in software release compared to traditional face-to-face teams (Herbsleb et al. 2001; Mockus et al. 2000).

> ***Proposition:*** Teams with members contributing at a higher level of on-task effort will be more effective.

- *Amount of knowledge and skill applied* also seem critical, though possibly difficult to measure and again perhaps not directly under the control of the project.

> ***Proposition:*** Teams with members who are more knowledgeable and skilled will be more effective.

- We will use coordination theory to analyze *task performance strategies*, as discussed below.

3.1.4 Moderating factors.

Finally, Hackman proposes factors that moderate the relationship between process and output, namely material resources, and between inputs and process, namely team synergy.

Material resources are things that the team requires in order to carry-out their task, for example computers, compliers and team collaboration systems, such as source code management tools (e.g. CVS) or more fully fledged collaborative environments, such as Sourceforge or Collab.net. However, since all of the groups we are studying seem to have access to the same sufficient resources this is not an aspect on which we intend to focus.

The review of software development presented above makes clear that practices for the development and maintenance of shared mental models will play an important role in enabling **team synergy**. We will apply collective mind [79] theory to conceptualize these models, as discussed below.

In the remainder of this section, we will discuss the two supporting theories we will use to extend Hackman's model, namely coordination theory and collective mind theory.

3.2 Coordination theory

We use coordination theory to analyze the structure of the tasks and coordination mechanisms used within teams. Many software process researchers have stressed the importance of coordination for software development [e.g., 16, 46]. For example, Kuwabara [47] states that, "coordination is a crucial element sustaining collective effort giving the Linux its integrity that unfolds the seemingly chaotic yet infinitely creative process of creation". The knowledge based-view of the firm [31] also emphasizes coordination mechanisms as important for integrating the knowledge of individuals into an organization's products, rules and routines.

Coordination theory provides a theoretical framework for analyzing coordination in processes. We use the model presented by Malone and Crowston [52], who define coordination as "managing dependencies." They analyzed processes in terms of actors performing interdependent tasks. These tasks might also require or create resources of various types. For example, in software development, developers

might require bug reports into order to create patches for the bugs. In this view, actors in organizations face coordination problems arising from interdependencies that constrain how tasks can be performed. Interdependencies can be between tasks, between tasks and the resources they need or between the resources used. Interdependencies may be inherent in the structure of the problem (e.g., components of a system may interact with each other, constraining how a particular component is designed [67]) or they may result from the assignment of tasks to actors and resources (e.g., two engineers working on the same component face constraints on the changes they can propose without interfering with each other). One implication of this view is that an important management strategy for software development work is to minimize dependencies, e.g., by creating software with modules that can be worked on independently.

Proposition: Teams with task structures and practices that minimize dependencies will be more effective.

To overcome the coordination problems created by dependencies, actors must perform additional work, which Malone and Crowston [52] called coordination mechanisms, or what Faraj and Xiao [21] call coordination practices. For example, if particular expertise is necessary to fix a bug (a task-actor dependency), then a developer with that expertise must be identified and the bug routed to him or her to work on. For any given dependency, there may be a range of available mechanisms, so project teams are expected to differ in their choice of mechanisms. It is unlikely that there is a single best set of mechanisms, but rather the fit of the selected mechanisms with other team practices is expected to have implications for effectiveness.

Proposition: Teams with coordination practices to manage dependencies will be more effective.

3.3 Collective mind

The second theory we apply is collective mind, a theory of the functioning of shared mental models. Shared mental models, as defined by Cannon-Bowers & Salas [6], "are knowledge structures held by members of a group that enable them to form accurate explanations and expectations for the task, and in turn, to coordinate their actions and adapt their behavior to demands of the task and other group members" (p. 228). Without shared mental models, individuals from different teams or backgrounds may interpret tasks differently, making collaboration and communication difficult [19] and diminishing individual contributions to the collective goal. Shared mental models are expected to lead to better team performance in general [6] and for software development in particular. Curtis, et al. [17], note that, "a fundamental problem in building large systems is the development of a common understanding of the requirements and design across the project group" (p. 52). They go on to say that, "transcripts of group meetings reveal the large amounts of time designers spend trying to develop a shared model of the design" (p. 52).

Following on work by Crowston and Kammerer [11], we intend to apply Weick and Robert's [79] collective mind theory to analyze this issue. We have chosen this theory for several reasons. First, previous conceptions of group mind have been controversial because they seemed to imply the existence of some super-individual entity [76]. By contrast, collective mind is described as an individual's "disposition to heed," hence an emphasis on "heedful" behaviors. If each member of a team has the desire and means to act in ways that further the goals and needs of the team (i.e., "heedfully"), then that team will exhibit behavior that might be described as collectively intelligent, even though it is the individuals who are intelligent, not the team *per se*. Second, Weick and Roberts [79] suggest that collective mind is beneficial for situations where there is need for high reliability, non-routine work, and interactive complexity (the combination of complex interactions with a high degree of coupling), all characteristics of much of software development. Finally, the elements of the theory fit cleanly into Hackman's model, as we now discuss.

Weick and Roberts [79] identify three overlapping individual behaviours that epitomize collective mind: 1) contribution (an individual member of a team contributes to the team outcome, one of Hackman's process factors), 2) representation (individuals build personal mental models of the team and its task, which we view as an important factor for Hackman's team synergy) and 3) subordination (an individual puts the team's goals ahead of individual goals, a team norm that corresponds to Hackman's team design input). Although conceptualized separately, these three concepts overlap and reinforce one another to some degree. For example, it is difficult to imagine heedful contributions from even highly talented and motivated individuals with weak representations of the team's needs and structure. While these actions go on in any group setting, the issue for collective mind is how carefully, appropriately and intelligently they are done. To the extent they are, the team will display collective mind.

Proposition: Teams with more highly developed shared mental models will be more effective.

Proposition: Teams in which members subordinate personal goals to team goals will be more effective.

Proposition: Teams with higher levels of socialization, conversation and narration will display more highly developed shared mental models.

4. PROPOSED DATA COLLECTION

In this section, we briefly describe the data we plan to collect to test the propositions presented in the previous section. As mentioned earlier, we are particularly interested in the practices adopted by effective development teams. Practices are often hard to study because they are taken for granted, and so escape intense observation. They go on all around us, but without notice unless something goes wrong. For on-line teams though, observation is facilitated because much of the team's interactions are funneled through a CMC system, and so structured and captured, as are the results of their work. Retrospective comparisons can be easily made by comparing data captured at different times, unbiased by the possibly selective recollections of informants. Our problem then is ensuring that these interactions present a complete picture of the team and then making sense of the vast pool of data created in the course of developers' interactions to answer interesting questions about their practices. To explore the concepts identified in the conceptual development section of this proposal, we will collect a wide range of data: project demographics, developer demographic data, interaction logs, code, project plans and procedures, as well as developer interviews, observation and participant observation. In the

Table 1. Summary of concepts in proposed model and corresponding phenomena.

Concepts	Specific phenomena
Team effectiveness	Code quality Project usage User satisfaction Project recognition
	Continued system development Group membership turnover
	Developer satisfaction Developer recognition
Team design	Task structure Process activities and dependencies Actors and roles
	Composition of team Experience Cross-membership
	Team norms about performance Socialization of new members
Process criteria	Number of developers Level of effort of developers (quantity and quality)
	Appropriate coordination mechanisms Team communication patterns
Team synergy	Shared mental models (representation) Socialization, narration, collaboration

remainder of this section, we will briefly review each source. Table 1 shows the mapping from each data source to analysis.

Project demographics. We plan to collect basic descriptive data about each project, such as its topic, intended use environment, programming language, etc. Often these data are self-reported by the developers to guide potential users (e.g., on SourceForge or FreshMeat, http://freshmeat.net/); in other cases, they can be inferred. We will also collect data indicative of the effectiveness of the project team [71], such as its level of activity, number of downloads as a proxy for use and development status, as well as any user ratings, such as FreshMeat user ratings. Again, SourceForge explicitly tracks these figures, but for other projects they may have to be inferred.

Developer demographic data. We will collect the list of developers for each project and their assigned roles, if any, plus any demographic information available. SourceForge collects skills ratings for a few developers; since only a minority of developers are rated at all, these are mostly interesting as a reflection of how well known a developer is. We also will collect developer's PGP or GnuPG key to examine the web of trust as a reflection of the developer's social network [60] (see http://www.chaosreigns.com/code/sig2dot/ for examples).

Developer interactions logs. The most voluminous source of data will be collected from archives of CMC tools used to support the team's interactions for FLOSS development work [39, 48]. These data are useful because they are unobtrusive measures of the team's behaviours [78]. Mailing list archives will be examined, as email is a primary tool used to support team coordination [14]. Such archives contain a huge amount of information: e.g., the Linux kernel list receives 5-7000 messages per month. From mailing lists, we will extract the date, sender and any individual recipient' names, the sender of the original message, in the case of a response, and text of each message. From bug tracking systems (e.g., Apache's GNATS, Linux kernel's Jitterbug, Mozilla's Bugzilla as well as Sourceforge's Tracker) we will extract data about bug typologies, who submitted bugs, who fixed them and the steps in the bug fixing process. We will examine features request archives and logs from other interaction tools, such as chat sessions.

Source code. A major advantage of studying open source software is that we have access to the source code itself. Many projects use a source code control system such as CVS, which stores intermediate versions of the source and the changes made. From these logs, we will be able to extract data on the kinds of contributions to understand the software structure and the date and name of the contributors to understand the role of individual developers [25, 32, 54]. Raw code poses numerous challenges to interpretation [72]. For example, not all projects assign authorship in the CVS tree. Again, we intend to leverage our analysis with work being carried out by other researchers [e.g., 42].

Project plans and procedures. Many projects have stated release plans and proposed changes. Such data are often available on the project's documentation web page or in a "status" file used to keep track of the agenda and working plans [15]. For example, Scacchi [66] examined requirements documentation for FLOSS projects. We will also examine any explicitly stated norms, procedures or rules for taking part in a project, such as the process to submit and handle bugs, patches or feature request. Such procedures are often reported on the project's web page (e.g., http://dev.apache.org /guidelines.html).

Developer attitudes and opinions. While the data sources listed above will provide an extensive pool of data, they are all indirect. Interviews and surveys are important to get rich, first-hand data about developers' perceptions and interpretations. We plan to conduct interviews with key informants in the selected projects. Interviews will be conducted mainly by e-mail, but we also plan to attend one or two FLOSS conferences each year (e.g., the *O'Reilly Open Source Convention* or *ApacheCon*) to interview FLOSS developers face-to-face. As part of the interviews protocol, we will employ the critical incident technique, in which developers are asked to describe personally experienced specific incidents which had an important effect on the final outcome [8]. We will also explore the developer's initial experiences of participation in FLOSS, the social structure and norms of the team, processes of knowledge exchange and socialization (especially the role of observation, which leaves no traces in the interaction logs), knowledge of other members' participation [57, 80] and impressions of project effectiveness. As well, interviews will be used to verify that the archives of interaction data give a fair and reasonably complete record of day-to-day interactions. In later phases of the project, a Web survey will be used to elicit attitudes and opinions from a large sample of developers.

Observation. We have found from an initial pilot study that developers interact extensively at conferences. Indeed, Nardi and Whittaker [58] note the importance of face-to-face interactions for sustaining social relations in distributed

teams. The FreeBSD developer Poul-Henning Kamp has also stated that phone calls can be occasionally used to solve complex problems [20]. These interactions are a small fraction of the total, but they may still be crucial to understanding the team's practices. We plan to use attendance at developer conferences as an opportunity to observe and document the role of face-to-face interaction for FLOSS teams.

We also intend to carry out a virtual ethnographic study of developer socialization and interaction. One student involved with the project has already virtually joined several development teams (with the permission of the project leaders and the knowledge of other members) and is currently participating in their normal activities while observing and recording these activities. In this way, we will study and learn first hand the socialization and coordination practices of these teams. We will track these teams through the various stages of development status, from planning through production/stable stage, observing how new members join the teams and how they contribute to the team output.

5. CONCLUSION

In this paper, we presented a conceptual model and a set of propositions concerning work practices within distributed FLOSS development teams. Developing a theoretical framework consolidating a number of theories to understand the dynamics within a distributed team is itself a contribution to the study of distributed teams and learning within organization literature [65].

Distributed work teams potentially provide several benefits but the separation between members of distributed teams creates difficulties in coordination, collaboration and learning, which may ultimately result in a failure of the team to be effective [2, 7, 41, 45]. Applying techniques from on information systems, distributed work and small groups research to software engineering will, we hope, allow us to better understand software engineering as a human-centered activity. Understanding the work practices of teams of software engineers working in a distributed environment is important to improve the effectiveness of distributed teams and of the traditional and non-traditional organizations within which they exist. The results of our study could serve as guidelines (in team governance, task coordination, communication practices, mentoring, etc.) to improve performance and foster innovation.

ABOUT THE AUTHORS

Our research team is based at the School of Information Studies at Syracuse University (Kevin Crowston, James Howison and Chengetai Masango) and the Information School at the University of Washington (Hala Annabi). Information Schools are interdisciplinary faculties researching information policy, information behavior, information management, information systems, information technology and information services.

This work was partially supported by NSF Grants 03-41475 and 04–14468.

REFERENCES

[1] Behlendorf, B. Open source as a business strategy. in Di Bona, C., Ockman, S. and Stone, M. eds. *Open sources: Voices from the open source revolution*, O'Reilly, San Francisco, 1999.

[2] Bélanger, F. and Collins, R. Distributed Work Arrangements: A Research Framework. *The Information Society*, *14* (2). 137–152.

[3] Bessen, J. Open Source Software: Free Provision of Complex Public Goods, Research on Innovation, 2002, 24 pages.

[4] Bezroukov, N. Open source software development as a special type of academic research (critique of vulgar raymondism). *First Monday*, *4* (10).

[5] Bezroukov, N. A second look at the Cathedral and the Bazaar. *First Monday*, *4* (12).

[6] Cannon-Bowers, J.A. and Salas, E. Reflections on shared cognition. *Journal of Organizational Behavior*, *22*. 195–202.

[7] Carmel, E. and Agarwal, R. Tactical approaches for alleviating distance in global software development. *IEEE Software* (March/April). 22–29.

[8] Chell, E. Critical incident technique. in Symon, G. ed. *Qualitative methods and analysis in organizational research: A practical guide*, Sage, London, 1998, 51–72.

[9] Cox, A. Cathedrals, Bazaars and the Town Council, 1998. Available from: http://slashdot.org /features/98/10/13/1423253.shtml. Accessed 22 March, 2004.

[10] Crowston, K., Annabi, H. and Howison, J. Defining Open Source Software project success. in *Proceedings of the 24th International Conference on Information Systems (ICIS 2003)*, Seattle, WA, 2003.

[11] Crowston, K. and Kammerer, E. Coordination and collective mind in software requirements development. *IBM Systems Journal*, *37* (2). 227–245.

[12] Crowston, K. and Scozzi, B. Open source software projects as virtual organizations: Competency rallying for software development. *IEE Proceedings Software*, *149* (1). 3–17.

[13] Cubranic, D., Open-source software development. in *2nd Workshop on Software Engineering over the Internet*, (Los Angeles, 1999).

[14] Cubranic, D. The ramp-up challenge in open-source software projects, Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, n.d.

[15] Cubranic, D. and Booth, K.S., Coordinating Open Source Software development. in *Proceedings of the 7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (1999).

[16] Curtis, B., Krasner, H. and Iscoe, N. A field study of the software design process for large systems. *CACM*, *31* (11). 1268–1287.

[17] Curtis, B., Walz, D. and Elam, J.J. Studying the process of software design teams. in *Proceedings of the 5th International Software Process Workshop On Experience With Software Process Models*, Kennebunkport, Maine, United States, 1990, 52–53.

[18] Di Bona, C., Ockman, S. and Stone, M. (eds.). *Open Sources: Voices from the Open Source Revolution*. O'Reilly & Associates, Sebastopol, CA, 1999.

[19] Dougherty, D. Interpretive barriers to successful product innovation in large firms. *Organization Science*, *3* (2). 179–202.

[20] Edwards, K., Epistemic communities, situated learning and Open Source Software development. in *Epistemic Cultures and the Practice of Interdisciplinarity Workshop*, (NTNU, Trondheim, 2001).

[21] Faraj, S. and Xiao, Y., Coordination in fast response organization. in *Academy of Management Conference*, (Denver, CO, 2002).
[22] Fielding, R.T. The Apache Group: A case study of Internet collaboration and virtual communities, 1997. Available from: http://www.ics.uci.edu/fielding/talks/ssapache /overview.htm.
[23] Franck, E. and Jungwirth, C. Reconciling investors and donators: The governance structure of open source, Lehrstuhl für Unternehmensführung und -politik, Universität Zürich, 2002.
[24] Gacek, C., Lawrie, T. and Arief, B. The many meanings of Open Source, Centre for Software Reliability, Department of Computing Science, University of Newcastle, Newcastle upon Tyne, United Kingdom, n.d.
[25] Gall, H., Hajek, K. and Jazayeri, M. Detection of Logical Coupling Based on Product Release History. in *Proceedings of the International Conference on Software Maintenance (ICSM '98)*, 1998.
[26] Gallivan, M.J. Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. *Information Systems Journal*, *11* (4). 277–304.
[27] Ghosh, R.A. Free/Libre and Open Source Software: Survey and Study. Report of the FLOSS Workshop on Advancing the Research Agenda on Free / Open Source Software, 2002. Available from: http://www.infonomics.nl/FLOSS /report/workshopreport.htm.
[28] Gladstein, D.L. Groups in context: A model of task group effectiveness. *Administrative Science Quarterly*, *29* (4). 499–517.
[29] Glass, R.L. Of open source, Linux, …and hype. *IEEE Software*, *16* (1). 126–128.
[30] Goodman, P.S., Ravlin, E.C. and Argote, L. Current thinking about groups: Setting the stage for new ideas. in Goodman, P.S. and Associates eds. *Designing Effective Work Groups*, Jossey-Bass, San Francisco, CA, 1986, 1–33.
[31] Grant, R.M. Prospering in dynamically-competitive environments: Organizational capability as knowledge integration. *Organizational Science*, *7* (4). 375–387.
[32] Graves, T.L. Inferring Change Effort from Configuration Management Databases, 1998.
[33] Guzzo, R.A. and Dickson, M.W. Teams in organizations: Recent research on performance effectiveness. *Annual Review of Psychology*, *47*. 307–338.
[34] Hackman, J.R. The design of work teams. in Lorsch, J.W. ed. *The Handbook of Organizational Behavior*, Prentice-Hall, Englewood Cliffs, NJ, 1986, 315–342.
[35] Hallen, J., Hammarqvist, A., Juhlin, F. and Chrigstrom, A. Linux in the workplace. *IEEE Software*, *16* (1). 52–57.
[36] Hann, I.-H., Roberts, J., Slaughter, S. and Fielding, R. Economic incentives for participating in open source software projects. in *Proceedings of the Twenty-Third International Conference on Information Systems*, 2002, 365–372.
[37] Hecker, F. Mozilla at one: A look back and ahead, 1999. Available from: http://www.mozilla.org/mozilla-at-one.html.
[38] Herbsleb, J.D. and Grinter, R.E. Splitting the organization and integrating the code: Conway's law revisited. in *Proceedings of the International Conference on Software Engineering (ICSE '99)*, ACM, Los Angeles, CA, 1999, 85–95.
[39] Herbsleb, J.D., Mockus, A., Finholt, T.A. and Grinter, R.E. An empirical study of global software development: Distance and speed. in *Proceedings of the International Conference on Software Engineering (ICSE 2001)*, Toronto, Canada, 2001, 81–90.
[40] Hertel, G., Niedner, S. and Herrmann, S. Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel, University of Kiel, Kiel, Germany, n.d., 39 pages.
[41] Jarvenpaa, S.L. and Leidner, D.E. Communication and trust in global virtual teams. *Organization Science*, *10* (6). 791–815.
[42] Koch, S. and Schneider, G. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, *12* (1). 27–42.
[43] Kogut, B. and Metiu, A. Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, *17* (2). 248–264.
[44] Kolodny, H. and Kiggundu, M. Towards the development of a sociotechnical systems model in Woodlands Mechanical Harvesting. *Human Relations*, *33*. 623–645.
[45] Kraut, R.E., Steinfield, C., Chan, A.P., Butler, B. and Hoag, A. Coordination and virtualization: The role of electronic networks and personal relationships. *Organization Science*, *10* (6). 722–740.
[46] Kraut, R.E. and Streeter, L.A. Coordination in software development. *Communications of the ACM*, *38* (3). 69–81.
[47] Kuwabara, K. Linux: A bazaar at the edge of chaos. *First Monday*, *5* (3).
[48] Lee, G.K. and Cole, R.E. The Linux Kernel Development As A Model of Open Source Knowledge Creation, Haas School of Business, University of California, Berkeley, Berkeley, CA, 2000.
[49] Leibovitch, E. The business case for Linux. *IEEE Software*, *16* (1). 40–44.
[50] Lerner, J. and Tirole, J. The open source movement: Key research questions. *European Economic Review*, *45*. 819–826.
[51] Ljungberg, J. Open Source Movements as a Model for Organizing. *European Journal of Information Systems*, *9* (4).
[52] Malone, T.W. and Crowston, K. The interdisciplinary study of coordination. *Computing Surveys*, *26* (1). 87–119.
[53] Markus, M.L., Manville, B. and Agres, E.C. What makes a virtual organization work? *Sloan Management Review*, *42* (1). 13–26.
[54] Mockus, A., Fielding, R.T. and Herbsleb, J.D. Two Case Studies Of Open Source Software Development: Apache And Mozilla. *ACM Transactions on Software Engineering and Methodology*, *11* (3). 309–346.
[55] Moody, G. *Rebel code—Inside Linux and the open source movement*. Perseus Publishing, Cambridge, MA, 2001.
[56] Moon, J.Y. and Sproull, L. Essence of distributed work: The case of Linux kernel. *First Monday*, *5* (11).
[57] Mortensen, M. and Hinds, P. Fuzzy teams: Boundary disagreement in distributed and collocated teams. in Hinds, P. and Kiesler, S. eds. *Distributed Work*, MIT Press, Cambridge, MA, 2002, 284–308.
[58] Nardi, B.A. and Whittaker, S. The place of face-to-face communication in distributed work. in Hinds, P. and Kiesler, S. eds. *Distributed Work*, MIT Press, Cambridge, MA, 2002, 83–110.
[59] Nieva, V.F., Fleshman, E.A. and Rieck, A. Team Dimensions: Their Identity, Their Measurement, and Their

Relationships, Advanced Research Resources Organizations, Washington, DC, 1978.
[60] O'Mahony, S. and Ferraro, F., Managing the Boundary of an 'Open' Project. in *Santa Fe Institute (SFI) Workshop on The Network Construction of Markets*, (2003).
[61] O'Reilly, T. Lessons from open source software development. *Communications of the ACM*, *42* (4). 33–37.
[62] Pfaff, B. Society and open source: Why open source software is better for society than proprietary closed source software, 1998. Available from: http://www.msu.edu/user/pfaffben/writings/anp/oss-is-better.html.
[63] Prasad, G.C. A hard look at Linux's claimed strengths…, n.d. Available from: http://www.osopinion.com /Opinions/GaneshCPrasad/GaneshCPrasad2-2.html.
[64] Raymond, E.S. The cathedral and the bazaar. *First Monday*, *3* (3).
[65] Robey, D., Khoo, H.M. and Powers, C. Situated-learning in cross-functional virtual teams. *IEEE Transactions on Professional Communication* (Feb/Mar). 51–66.
[66] Scacchi, W. Understanding the requirements for developing Open Source Software systems. *IEE Proceedings Software*, *149* (1). 24–39.
[67] Schach, S.R., Jin, B., Wright, D.R., Heller, G.Z. and Offutt, A.J. Maintainability of the Linux Kernel, 2003. Available from: http://www.vuse.vanderbilt.edu /%7Esrs/preprints/linux.longitudinal.preprint.pdf. Accessed 14 Dec, 2003.
[68] Shepard, T., Lamb, M. and Kelly, D. More testing should be taught. *Communication of the ACM*, *44* (6). 103–108.
[69] Stamelos, I., Angelis, L., Oikonomou, A. and Bleris, G.L. Code quality analysis in open source software development. *Information Systems Journal*, *12* (1). 43–60.
[70] Stewart, K.J. and Ammeter, T. An exploratory study of factors influencing the level of vitality and popularity of open source projects. in *Proceedings of the Twenty-Third International Conference on Information Systems*, 2002, 853–857.
[71] Stewart, K.J. and Gosain, S., Impacts of ideology, trust, and communication on effectivness in open source software development teams. in *Twenty-Second International Conference on Information Systems*, (New Orleans, LA, 2001), 507–512.
[72] Tuomi, I. Evolution of the Linux Credits File: Methodological Challenges and Reference Data for Open Source Research, 2002. Available from: http://www.jrc.es/~tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf. Accessed 15 November, 2002.
[73] Valloppillil, V. Halloween I: Open Source Software, 1998. Available from: http://www.opensource.org/halloween /halloween1.html.
[74] Valloppillil, V. and Cohen, J. Halloween II: Linux OS Competitive Analysis, 1998. Available from: http://www.opensource.org/halloween/halloween2.html.
[75] Vixie, P. Software engineering. in Di Bona, C., Ockman, S. and Stone, M. eds. *Open sources: Voices from the open source revolution*, O'Reilly, San Francisco, 1999.
[76] Walsh, J.P. Managerial and organizational cognition: Notes from a trip down memory lane. *Organization Science*, *6* (3). 280–321.
[77] Wayner, P. *Free For All*. HarperCollins, New York, 2000.
[78] Webb, E. and Weick, K.E. Unobtrusive measures in organizational theory: A reminder. *Administrative Science Quarterly*, *24* (4). 650–659.
[79] Weick, K.E. and Roberts, K. Collective mind in organizations: Heedful interrelating on flight decks. *Administrative Science Quarterly*, *38* (3). 357–381.
[80] Weisband, S. Maintaining awareness in distributed team collaboration: Implications for leadership and performance. in Hinds, P. and Kiesler, S. eds. *Distributed Work*, MIT Press, Cambridge, MA, 2002, 311–333.
[81] Young, R. How Red Hat Software stumbled across a new economy model and helped improve an industry. in Di Bona, C., Ockman, S. and Stone, M. eds. *Open sources: voices from the open source revolution*, O'Reilly, San Francisco, 1999.