2005

Effective Work Practices for FLOSS Development: A Model and Propositions

Kevin Crowston
Syracuse University

Hala Annabi
University of Washington

James Howison
Syracuse University

Chengetai Masango
Syracuse University

Recommended Citation

Crowston, K., Annabi, H., Howison, J. & Masango, C. Effective work practices for FLOSS development: A model and propositions. In Proceedings of the Thirty-Eighth Hawai'i International Conference on System Science (HICSS–38). Kona, HI, USA, January. doi: 10.1109/HICSS.2005.222

Effective work practices for FLOSS development: A model and propositions[1]

Kevin Crowston[*], Hala Annabi[**], James Howison[*] and Chengetai Masango[*]
[*] *Syracuse University School of Information Studies*
crowston@syr.edu, jhowison@syr.edu, cmasango@syr.edu

[**] *University of Washington, The Information School*
hpannabi@u.washington.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 [1] model of effectiveness of work teams, with coordination theory [2] and collective mind [3] 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.

1. Introduction

Free/Libre Open Source Software (FLOSS)[2] 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. 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).

Key to our interest 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) [4, 5]. 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 [6]. As well, many (though by no means all) programmers contribute to projects as volunteers, without working for a common organization or being paid. This heavy reliance on self-organization sets FLOSS teams apart from most other distributed teams.

In this paper, we review the literature on FLOSS development and distributed software development more generally. We then develop a theoretical model to explain the performance of FLOSS teams drawing on research on group work. We use the model to propose a set of propositions to guide further research.

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., 7, 8, 9]. Second, various explanations have been proposed for the decision by individuals to contribute to projects without pay [e.g., 10, 11-14]. These authors have mentioned factors such as personal interest, ideological commitment, development of skills [15] or enhancement of reputation [14]. Finally, a few authors have investigated the processes of FLOSS development [e.g., 4, 16], which is the focus of this paper.

Raymond's [4] bazaar metaphor is the most well-known model of the FLOSS process. While popular, the bazaar metaphor has been broadly criticized. According to

[1] This research was partially supported by NSF Grants 03-41475 and 04–14468. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[2] FLOSS 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 accomodate this range of meanings.

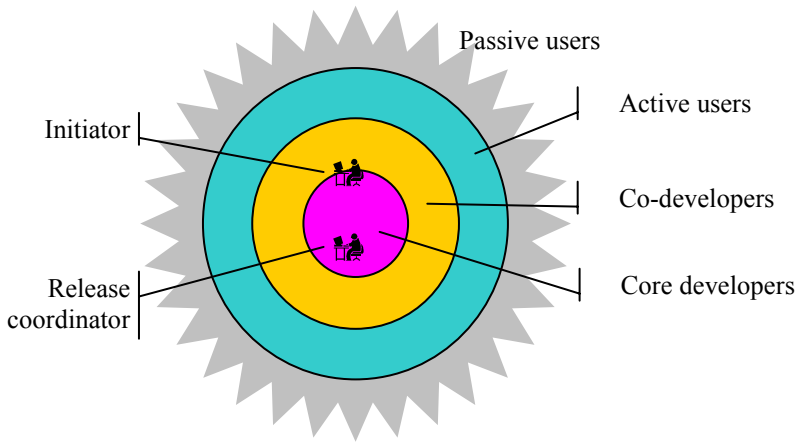

Figure 1. Hypothesized FLOSS development team structure.

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 [17, 18].

Recent empirical work has begun to illuminate the structure and function of FLOSS development teams. Gallivan [19] 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 [20, 21], 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 [22-24]. This barrier is particularly troubling because of the reliance of FLOSS projects on volunteer submissions and "fresh blood" [25]. 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., [26]). 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 [27-33]. 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 [34, 35], 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 [36]. (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 [37]. 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 [38]. The FLOSS process enables hundreds of people to work on these parts of the process [39]. 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 [39].

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 [40]. There are only a few well-documented case studies [e.g., 41], most of which discuss successes rather than failures. Finally, with a few exceptions [e.g., 42, 43], 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 [44]. Our work is intended to fill some of these gaps by providing a theoretically-based model of FLOSS development practices.

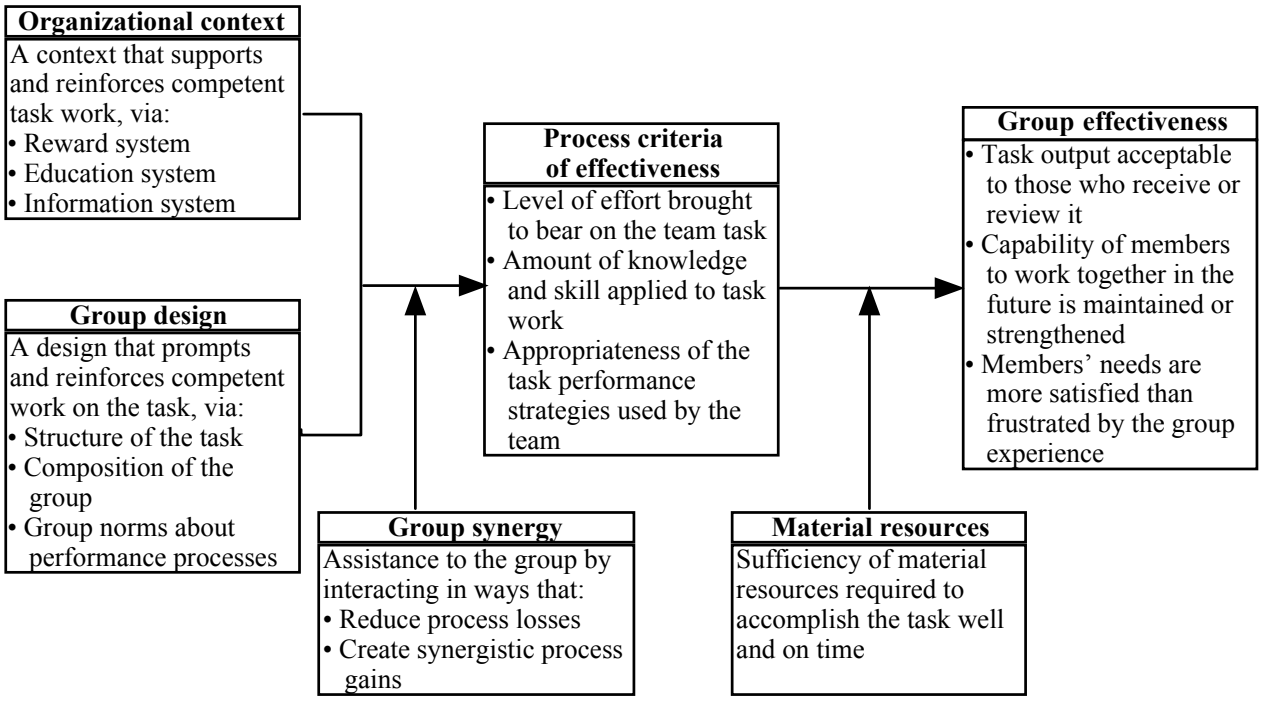

Figure 2. Hackman's [1] normative model of group effectiveness.

3. Theory

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 [45, 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 [1] 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 [46], we use coordination theory [2] and collective mind [3] 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 [1], shown in Figure 2. Hackman's [1] model is broadly similar to other models [47], such as [48], [49] or [50]. 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 [1] 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 [1] 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 [51], 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.

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

3.1.2 Inputs. Hackman's model includes two sets of **input** factors, *organizational context* and *group design*. Organizational context includes three factors:

- a reward system that provides challenging objectives and consequences for excellent performance and thus motivates effort;
- an educational system that provides outside expertise to support appropriate knowledge and skills; and
- an information system that provides information about the situation and likely outcomes of alternative actions to enable appropriate task strategies.

For FLOSS teams though, identifying the organizational context is problematic because teams are generally composed of individuals from multiple organizations and contexts. This diversity may be advantageous, e.g., if the team can take advantage of expertise available in different settings. Alternately, it can be argued that the broader FLOSS community itself provides the context, e.g., by rewarding contributors with recognition. In either case, these systems would not be under the control of projects. However, to the extent that FLOSS teams are self-organized, we argue that teams can create their own organizational contexts. In particular, we propose:

> ***Proposition:*** Teams with practices that set challenging but obtainable goals will be more effective.

> ***Proposition:*** Teams with practices that reward members for contribution will be more effective.

> ***Proposition:*** Teams with practices that access outside expertise will be more effective.

> ***Proposition:*** Teams with practices that gather information about the situation and alternative actions will be more effective.

The next set of inputs is *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 differences in the way teams *structure the task*. For example, Harter et al. [52] found that the maturity of the software process was related to development quality. Some differences may relate to differences in the complexity, uncertainty and scope of the software being developed. To analyze task structure, 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 on FLOSS 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 a variety of 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*" [1].

- Prior work has noted that distributed teams often need to expend more *effort* to be effective [53], suggesting the importance of the level of effort in the process. Effort is important both individually and collectively. An important factor for the success of FLOSS teams is their ability to attract developers.

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

> ***Proposition:*** Teams with practices to attract contributions from more developers will be more effective.

> ***Proposition:*** Teams with practices to attract contributions from more active users 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**.

For software development, relevant **material resources** would seem to be limited to development tools, which are readily available, thanks to systems like SourceForge (http://sourceforge.net/) and Savannah (http://savannah .gnu.org/), which host thousands of projects. Therefore, we do not include this factor in our current theorizing. For future research, we plan to look for ways in which tool use structures team practices.

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 [3] 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., 36, 54]. For example, Kuwabara [55] 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 [56] 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 [2], 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 [57]) 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 [2] called coordination mechanisms, or what Faraj and Xiao [58] 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 that to occur teams must have collective mind as discussed in the next section. 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 practices that 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 [59], "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 [60] and diminishing individual contributions to the collective goal.

Shared mental models are expected to lead to better team performance in general [59] and for software development in particular. Curtis, et al. [61], 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).

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

We note though that FLOSS teams are hypothesized to have members contributing in a variety of roles, and shared mental models are likely more important for a core member than for a peripheral member. As well, the need for shared mental models may be reduced if there are fewer dependencies among the tasks being performed.

Following on work by Crowston and Kammerer [46], we intend to apply Weick and Robert's [3] collective mind theory to analyze shared mental models. 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 [62]. 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 [3] 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.

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
Organizational context	Practices that set goals and reward contributions
	Practices that access outside knowledge
	Practices that access information about task and alternatives
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

Weick and Roberts [3] 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). We note though that membership in FLOSS teams is generally voluntary, meaning that teams may not be able to demand subordination from team members. They may instead rely on alignment between personal and collective goals, which is closely related to the development of an effective project reward system.

> ***Proposition:*** Teams with practices that align individual members' goals and team goals will be more effective.

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.

Given the importance of collective mind, we will look not only for practices that exhibit it, but also those that build and maintain it. For the later purpose, Brown and Duguid's [63] model of communities of practice seems

useful. Brown and Duguid [63] suggested three overlapping social processes that underlie work practices: social construction, narration, and collaboration. Construction (or socialization) addresses the issue of people joining a team needing to understand how they fit into the process being performed (i.e., their representation, contribution and subordination). New members need to be encouraged and educated to interact with one another to develop a strong sense of "how we do things around here" (i.e., representation) [64]. Second, Brown and Duguid [63] stress the importance of narration. To keep the collective mind strong and viable, important events must be "replayed" and reanalyzed, and the history that defines who the group is and how it does things (representation) must be continually reinforced, reinterpreted, and updated and shared with newcomer. Because the teams do not meet face-to-face regular, opportunities for this type of interaction may have to be deliberately created. Finally, Brown and Duguid [63] stress the importance of collaboration, based on narration, thus leading to the team synergy identified in Hackman's model.

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

Table 1 summarizes the constructs we will explore in future studies of FLOSS development using this model.

4. 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].

We are currently applying the model in a field study of FLOSS teams. To ground the concepts developed above, we are collecting a wide variety of evidence, including logs of ICT-supported interactions, bug reports, code changes and project documents, as well as interviews with developers. These data will be analyzed primarily through content analysis, but also by creating process maps, cognitive maps and social networks.

Understanding the work practices of teams of independent knowledge workers 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. Distributed work teams potentially provide several benefits but the separation between members of distributed teams creates difficulties in coordination and collaboration, which may ultimately result in a failure of the team to be effective [66-69].

5. References

[1] J. R. Hackman, "The design of work teams," in *The Handbook of Organizational Behavior*, J. W. Lorsch, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1986, pp. 315–342.

[2] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," *Computing Surveys*, vol. 26, pp. 87–119, 1994.

[3] K. E. Weick and K. Roberts, "Collective mind in organizations: Heedful interrelating on flight decks," *Administrative Science Quarterly*, vol. 38, pp. 357–381, 1993.

[4] E. S. Raymond, "The cathedral and the bazaar," *First Monday*, vol. 3, 1998.

[5] P. Wayner, *Free For All*. New York: HarperCollins, 2000.

[6] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," in *Proceedings of the International Conference on Software Engineering (ICSE '99)*. Los Angeles, CA: ACM, 1999, pp. 85–95.

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

[8] B. Kogut and A. Metiu, "Open-source software development and distributed innovation," *Oxford Review of Economic Policy*, vol. 17, pp. 248–264, 2001.

[9] J. Lerner and J. Tirole, "The open source movement: Key research questions," *European Economic Review*, vol. 45, pp. 819–826, 2001.

[10] G. Hertel, S. Niedner, and S. Herrmann, "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.

[11] I.-H. Hann, J. Roberts, S. Slaughter, and R. Fielding, "Economic incentives for participating in open source software projects," in *Proceedings of the Twenty-Third International Conference on Information Systems*, 2002, pp. 365–372.

[12] J. Bessen, "Open Source Software: Free Provision of Complex Public Goods," Research on Innovation July 2002.

[13] E. Franck and C. Jungwirth, "Reconciling investors and donators: The governance structure of open source," Lehrstuhl für Unternehmensführung und -politik, Universität Zürich, Working Paper No. 8, June 2002.

[14] M. L. Markus, B. Manville, and E. C. Agres, "What makes a virtual organization work?," *Sloan Management Review*, vol. 42, pp. 13–26, 2000.

[15] J. Ljungberg, "Open Source Movements as a Model for Organizing," *European Journal of Information Systems*, vol. 9, 2000.

[16] K. J. Stewart and T. Ammeter, "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, pp. 853–857.

[17] N. Bezroukov, "Open source software development as a special type of academic research (critique of vulgar raymondism)," *First Monday*, vol. 4, 1999.

[18] N. Bezroukov, "A second look at the Cathedral and the Bazaar," *First Monday*, vol. 4, 1999.

[19] M. J. Gallivan, "Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies," *Information Systems Journal*, vol. 11, pp. 277–304, 2001.

[20] J. Y. Moon and L. Sproull, "Essence of distributed work: The case of Linux kernel," *First Monday*, vol. 5, 2000.

[21] A. Cox, "Cathedrals, Bazaars and the Town Council," http://slashdot.org/features/98/10/13/1423253.shtml, 1998, accessed 22 March 2004.

[22] R. T. Fielding, "The Apache Group: A case study of Internet collaboration and virtual communities," http://www.ics.uci.edu/fielding/talks/ssapache/overview.htm., 1997.

[23] C. Gacek and B. Arief, "The many meanings of Open Source," *IEEE Software*, vol. 21, pp. 34–40, 2004.

[24] F. Hecker, "Mozilla at one: A look back and ahead," http://www.mozilla.org/mozilla-at-one.html, 1999.

[25] D. Cubranic and K. S. Booth, "Coordinating Open Source Software development," presented at Proceedings of the 7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999.

[26] K. J. Stewart and S. Gosain, "Impacts of ideology, trust, and communication on effectivness in open source software development teams," presented at Twenty-Second International Conference on Information Systems, New Orleans, LA, 2001.

[27] V. Valloppillil, "Halloween I: Open Source Software," http://www.opensource.org/halloween/halloween1.html, 1998.

[28] K. Crowston and B. Scozzi, "Open source software projects as virtual organizations: Competency rallying for software development," *IEE Proceedings Software*, vol. 149, pp. 3–17, 2002.

[29] G. C. Prasad, "A hard look at Linux's claimed strengths…," http://www.osopinion.com/Opinions/ GaneshCPrasad/GaneshCPrasad2-2.html, n.d.

[30] V. Valloppillil and J. Cohen, "Halloween II: Linux OS Competitive Analysis," http://www.opensource.org /halloween/halloween2.html, 1998.

[31] J. Hallen, A. Hammarqvist, F. Juhlin, and A. Chrigstrom, "Linux in the workplace," *IEEE Software*, vol. 16, pp. 52–57, 1999.

[32] E. Leibovitch, "The business case for Linux," *IEEE Software*, vol. 16, pp. 40–44, 1999.

[33] B. Pfaff, "Society and open source: Why open source software is better for society than proprietary closed source software," http://www.msu.edu/user/pfaffben/writings/anp /oss-is-better.html, 1998.

[34] G. Moody, *Rebel code—Inside Linux and the open source movement*. Cambridge, MA: Perseus Publishing, 2001.

[35] P. Vixie, "Software engineering," in *Open sources: Voices from the open source revolution*, C. Di Bona, S. Ockman, and M. Stone, Eds. San Francisco: O'Reilly, 1999.

[36] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Communications of the ACM*, vol. 38, pp. 69–81, 1995.

[37] T. O'Reilly, "Lessons from open source software development," *Communications of the ACM*, vol. 42, pp. 33–37, 1999.

[38] T. Shepard, M. Lamb, and D. Kelly, "More testing should be taught," *Communication of the ACM*, vol. 44, pp. 103–108, 2001.

[39] G. K. Lee and R. E. Cole, "The Linux Kernel Development As A Model of Open Source Knowledge Creation," Haas School of Business, University of California, Berkeley, Berkeley, CA, Unpublished manuscript December 2000 2000.

[40] R. L. Glass, "Of open source, Linux, …and hype," *IEEE Software*, vol. 16, pp. 126–128, 1999.

[41] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two Case Studies Of Open Source Software Development: Apache And Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 309–346, 2002.

[42] R. Young, "How Red Hat Software stumbled across a new economy model and helped improve an industry," in *Open sources: voices from the open source revolution*, C. Di Bona, S. Ockman, and M. Stone, Eds. San Francisco: O'Reilly, 1999.

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

[44] D. Cubranic, "Open-source software development," presented at 2nd Workshop on Software Engineering over the Internet, Los Angeles, 1999.

[45] R. A. Guzzo and M. W. Dickson, "Teams in organizations: Recent research on performance effectiveness," *Annual Review of Psychology*, vol. 47, pp. 307–338, 1996.

[46] K. Crowston and E. Kammerer, "Coordination and collective mind in software requirements development," *IBM Systems Journal*, vol. 37, pp. 227–245, 1998.

[47] P. S. Goodman, E. C. Ravlin, and L. Argote, "Current thinking about groups: Setting the stage for new ideas," in *Designing Effective Work Groups*, P. S. Goodman and Associates, Eds. San Francisco, CA: Jossey-Bass, 1986, pp. 1–33.

[48] H. Kolodny and M. Kiggundu, "Towards the development of a sociotechnical systems model in Woodlands Mechanical Harvesting," *Human Relations*, vol. 33, pp. 623–645, 1980.

[49] D. L. Gladstein, "Groups in context: A model of task group effectiveness," *Administrative Science Quarterly*, vol. 29, pp. 499–517, 1984.

[50] V. F. Nieva, E. A. Fleshman, and A. Rieck, "Team Dimensions: Their Identity, Their Measurement, and Their Relationships," Advanced Research Resources Organizations, Washington, DC, Final Technical Report for Contract No. DAHC19-78-C-0001 1978.

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

[52] D. E. Harter and S. Slaughter, "Process maturity and software quality: A field study," in *Proceedings of the Twenty-First International Conference on Information Systems*, S. Ang, H. Krcmar, W. J. Orlikowski, P. Weill, and J. I. DeGross, Eds. Brisbane, Australia, 2000, pp. 407–411.

[53] R. J. Ocker and J. Fjermestad, "High versus low performing virtual design teams: A preliminary analysis of communication," in *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000, pp. 10 pages.

[54] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *CACM*, vol. 31, pp. 1268–1287, 1988.

[55] K. Kuwabara, "Linux: A bazaar at the edge of chaos," *First Monday*, vol. 5, 2000.

[56] R. M. Grant, "Prospering in dynamically-competitive environments: Organizational capability as knowledge integration," *Organizational Science*, vol. 7, pp. 375–387, 1996.

[57] S. R. Schach, B. Jin, D. R. Wright, G. Z. Heller, and A. J. Offutt, "Maintainability of the Linux Kernel," Department of Electrical Engineering and Computer Science, Vanderbilt University, http://www.vuse.vanderbilt.edu/%7Esrs/preprints/linux.longitudinal.preprint.pdf, 2003, accessed 14 Dec 2003.

[58] S. Faraj and Y. Xiao, "Coordination in fast response organization," presented at Academy of Management Conference, Denver, CO, 2002.

[59] J. A. Cannon-Bowers and E. Salas, "Reflections on shared cognition," *Journal of Organizational Behavior*, vol. 22, pp. 195–202, 2001.

[60] D. Dougherty, "Interpretive barriers to successful product innovation in large firms," *Organization Science*, vol. 3, pp. 179–202, 1992.

[61] B. Curtis, D. Walz, and J. J. Elam, "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, pp. 52–53.

[62] J. P. Walsh, "Managerial and organizational cognition: Notes from a trip down memory lane," *Organization Science*, vol. 6, pp. 280–321, 1995.

[63] J. S. Brown and P. Duguid, "Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation," *Organization Science*, vol. 2, pp. 40–57, 1991.

[64] M. O'Leary, W. J. Orlikowski, and J. Yates, "Distributed work over the centuries: Trust and control in the Hudson's Bay Company, 1670–1826," in *Distributed Work*, P. Hinds and S. Kiesler, Eds. Cambridge, MA: MIT Press, 2002, pp. 27–54.

[65] D. Robey, H. M. Khoo, and C. Powers, "Situated-learning in cross-functional virtual teams," *IEEE Transactions on Professional Communication*, pp. 51–66, 2000.

[66] S. L. Jarvenpaa and D. E. Leidner, "Communication and trust in global virtual teams," *Organization Science*, vol. 10, pp. 791–815, 1999.

[67] F. Bélanger and R. Collins, "Distributed Work Arrangements: A Research Framework," *The Information Society*, vol. 14, pp. 137–152, 1998.

[68] R. E. Kraut, C. Steinfield, A. P. Chan, B. Butler, and A. Hoag, "Coordination and virtualization: The role of electronic networks and personal relationships," *Organization Science*, vol. 10, pp. 722–740, 1999.

[69] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Software*, pp. 22–29, 2001.