Submission to ICIS 2009 Dissertation Competition

LAYERED COLLABORATION:

A SOCIO-TECHNICAL THEORY OF MOTIVATION AND ORGANIZATION FOR FREE AND OPEN SOURCE SOFTWARE DEVELOPMENT

James Howison
Syracuse University
james@howison.name

LAYERED COLLABORATION:

A SOCIO-TECHNICAL THEORY OF MOTIVATION AND ORGANIZATION FOR FREE AND OPEN SOURCE SOFTWARE DEVELOPMENT

James Howison
Syracuse University
james@howison.name

Abstract

This paper develops a theory of layered collaboration in the development of Free and Open Source Software (FLOSS). Through a research arc of discovery (participant observation), replication (two archival case studies) and formalization (a model of developer choices) it is argued that work is far more often individual work done "in company" than it is teamwork. When tasks appear too large for an individual they are more likely to be deferred until they are easier rather than be undertaken through teamwork. It is theorized that this way of organizing is successful because it fits with the motivations of the participants, the nature of software as an IT artifact and its development as a task, and the key technologies of FLOSS collaboration. The theory, and the empirical findings in which it is grounded, are important because they enable a systematic approach to understanding the implications of FLOSS development as a model for adaptation and the future of informationalized work. Accordingly, the discussion enumerates the conditions under which this theory of organizing is likely to be successful, many stemming directly from software, an IT artifact, as an object of collaboration such as non-revocable and rewindable work with incremental incentives. These are used as a framework to analyze efforts to adapt the FLOSS model of organizing for self-organizing, virtual teams in other domains of work.

Introduction

What if you had to organize to not only do something great, but also to attract the very people you need to do those great things, without any money at all? How might that organizing—that way of working—look? A great deal of literature has studied the more common case of organization in which the future availability of resources is assumed: teams are studied working, but only after members are assigned; partnerships between businesses are studied after their members are engaged. Typically this inquiry rests on the knowledge that participants are paid for their involvement and, *quid pro quo*, they offer their working time as a resource for an active management pursuing a goal of organizational efficiency.

Increasingly, however, the future of work seems unlikely to look like this, particularly with the growth of virtual teams and distributed work supported by information systems. Simply offering money and then telling people what to do isn't always enough to attract the best and brightest, and even if they are attracted it might not be enough to hold onto them or to ensure that they do their best work. Skilled workers, especially in-demand knowledge workers, might better be understood and managed if they were perceived as acting like volunteers (e.g., Barnard et al. 1968; Cook 2008; Drucker 2002; Handy 1988; Malone 2004).

This paper examines the question of the impact of motivations on the organization of virtual work, particularly work where the participants are distributed volunteers. This question is doubly important to the field of Information Systems. Firstly, recently new phenomena with these characteristics, such as free and open source software development and Wikipedia, have emerged to great acclaim, each supported by information systems. Secondly, it is hoped that traditional business organizations, and particularly the IS function, can learn from these phenomena. These questions are doubly important because the phenomena offer potentially alternative methods of accomplishing

work traditionally considered part of the IS function (software development and knowledge management) and they are occurring *via* innovative Information Systems.

Specifically this paper focuses on free (libre) and open source software development, herein abbreviated as FLOSS. FLOSS is a canonical type of distributed, online production (e.g.,von Hippel & von Krogh, 2003; McLure Wasko & Faraj, 2005). The projects studied in this paper are community-based FLOSS projects, meaning that they have no institutional existence (e.g. non-profit foundations) nor do they have significant corporate involvement; in this way the projects chosen epitomize what is most novel in the FLOSS phenomenon.

Existing research on FLOSS development has concentrated in three areas: inputs (including motivations), processes (such as coordination or governance) and outputs (such as implementations or software quality). It is established that FLOSS participants are driven by a variety of motivations (e.g., Feller et al., 2005; Ghosh et al., 2002; Lakhani & Wolf, 2003; Lakhani & von Hippel, 2003), including the need for software itself, learning, ideological commitment and, particularly for long-term participants, reputation, although not to the degree suggested by economists (e.g., Lerner & Tirole, 2002). Process research has focused on coordination, documenting the prevalence of practices such as self-assignment of tasks (the essence of volunteerism), "short-cut" decision making processes as well as a tendency to "code first" and then work together on integration (e.g., Crowston et al., 2005; Yamauchi et al., 2000). This paper, and the dissertation it is drawn from, contribute to this literature by linking motivation and process, arguing that by looking at these together one can understand the FLOSS phenomenon better.

Research on the organization of work has paid particular attention to the concept of interdependence. A strong line of literature has argued that structures embedded in the work itself determine the type of organization most suited to that work (e.g., Thompson, 1967; Van de Ven et al., 1976; Malone & Crowston, 1994). Recently, however, this line of thinking has been challenged by studies driven by a practice view of work (e.g. Shea & Guzzo, 1987; Wageman, 1995; Wageman & Gordon, 2005). The challenge suggests that similar work can be accomplished well with very different patterns of interdependence, and moreover that patterns of appropriate interdependence are driven as much by factors such as individual preferences, task definitions and technological affordances as by unchangeable requirements of work. This dissertation attends to a gap in this literature by introducing the impact of participant motivations on patterns of interdependence and work. It argues that they are important even when building software collaboratively, a task where the technical interdependence of the work has been held to be especially important (e.g., Herbsleb et al., 2001).

The purpose of this dissertation is theory development, which is undertaken though an unfolding arc of discovery, replication and formalization. Discovery consisted of four years of participant observation in a community-based FLOSS project, while replication consisted of an archive-based field study in two similar FLOSS projects. Finally the theory is elaborated and formalized through a rational expectations model of developer decision-making. Overall the dissertation provides a framework to discuss the challenges of adapting FLOSS organization for other types of work and institutional environments, including the IS function in traditional, for-profit businesses.

Discovery: Participant Observation

2003 email (on discussing a desirable feature):
I really want to use this, but the conditions have never quite been right - either I was waiting for ... RSS+RDF (now looks like it'll never happen) or ... an XML bibliographic file format ... (could happen now, but I ran out of free time).

2007 email (on checking in working code for this feature):
It was much easier than I expected it to be because the existing groups code (and search groups code) was very easy to extend. Kudos - I wouldn't have tried it if so much hadn't already been solved well.

Figure 1. An episode from BibDesk, discussing the ability to subscribe to online reference lists. These two emails were sent four years apart: the complex work was deferred until other work, done for its own purposes, had made the original desired feature possible to accomplish through short, individual work.

The author participated in and observed the BibDesk project over a four-year period. BibDesk is a community-based FLOSS project to build a reference manager akin to Endnote. In particular two aspects of organization that had not been considered in the existing literature particularly stood out: work (both mine and that of others) was accomplished primarily through short, individual tasks that built on previous work, while complex work was

deferred indefinitely, only being undertaken when a changing codebase had made it easier. Figure 1 illustrates these aspects through two emails from the project founder (not the researcher), written four years apart.

Introspection, afforded by participant observation, on similar episodes experienced by the author pointed to the involvement of two aspects of volunteer motivation. Firstly I was only able to work on the project in my "free time" and my free time did not come consistently, therefore I was not keen to take on tasks that I did not feel I could finish in the time I knew I had available. Secondly, I worked alone because I did not want to rely on the free time and commitment of others to finish my work; I felt I had "no right" to request their time and since their commitment was unpredictable I could not rely on their portion of shared work being completed.

Replication: Archival Case Studies

In order to deepen, strengthen and challenge the insights gained from participant observation, I undertook an archive-based field study. The specific goal of the study was to see whether the findings from the participant observation, specifically the layering of short, individual episodes of work and the deferral of complex work, could be replicated; for this reason cases similar to BibDesk and to each other were selected: Fire and Gaim, both community-based instant messaging clients hosted on Sourceforge.

Participant observation had indicated that work proceeds, and is inscribed in, many project venues, and a coherent understanding of the project's organization could not be obtained from single venues, such as the mailing list, alone. Therefore data was collected to be as comprehensive as possible: Mailing lists, Source code repositories (CVS and SVN), Forums, Issue Trackers and Release Notes. Participant observation identified the period between two releases as a natural and meaningful periodization of the projects' work life, so, using the concepts above, I analyzed an inter-release period (approximately 45 days) for each project, chosen to be close in calendar time so the projects work would be dealing with similar external contexts.

I developed a set of concepts for organizing the work of the projects. I defined a Task Outcome as a change to the shared outputs of the project, usually the software but also including documentation or the project website. A Task, then, was a series of Actions undertaken by Participants contributing to the Task Outcome. Documents provide archival evidence of these Actions.

I began with the release notes and the README file, which evolve over the period as the participant's own categorization of their work by Task Outcome. I then worked to assign Documents relevant to each Task, working iteratively through the full set of project archives. I worked iteratively until I had exhausted the records from the source code repository (since they all alter the shared work product). I then worked through the relevant Documents, identifying Actions, as well as ascertaining which Participant performed the Action. These Actions were then classified, according to their contribution to the Task, as: 1) Management, 2) Review, 3) Production, 4) Documentation and 5) Support (primarily bug reports or feature requests, answering questions about code or testing code). At this point the dataset was ready for analysis to see if the participant observation results could be replicated.

I found clear evidence replicating the finding of work being undertaken in short and individual episodes. As shown in Figure 2, I found that ~80% of the 106 Tasks involved only a single participant writing code, a further ~10% of tasks were primarily programmed by a single participant, with a small amount of 'polishing' work, such as fixing a spelling mistake by another participant, and less than ~10% involved more than programmer, (these I called Co-work). Even within the Co-work episodes only one involved planning Actions synchronizing the work of two programmers. All Tasks were generally short, with both the mean and median duration being shorter than 1 week (longer outliers are discussed below). The Co-work Tasks showed no signs of systematically greater complexity, such as involving more lines of code.

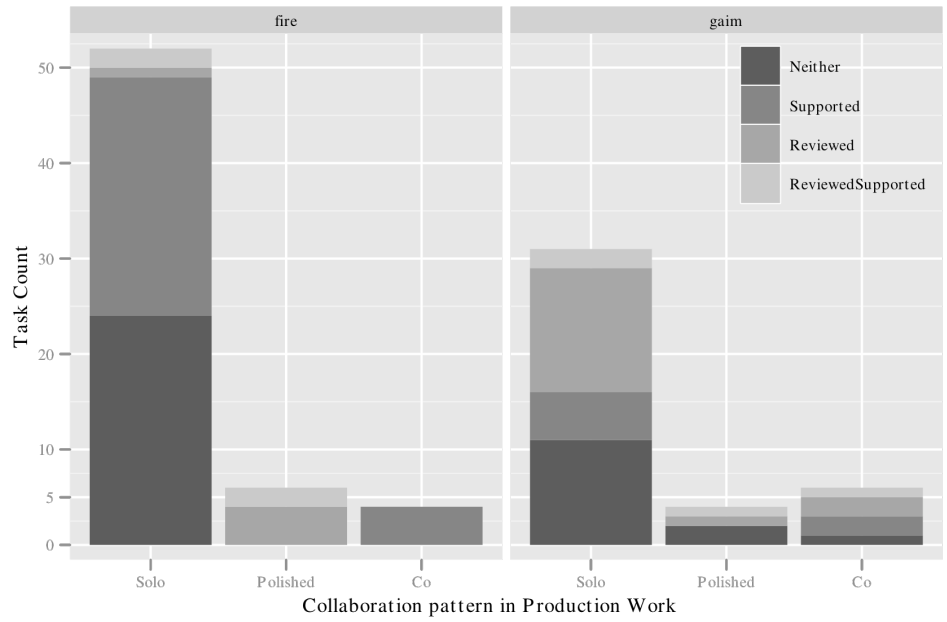

Figure 2. Tasks tended to involve only individual programmers. Figure shows tasks in Fire and Gaim classified primarily by the number of programmers (polished work involved a single main programmer; others did only polishing work such as fixing a typo).

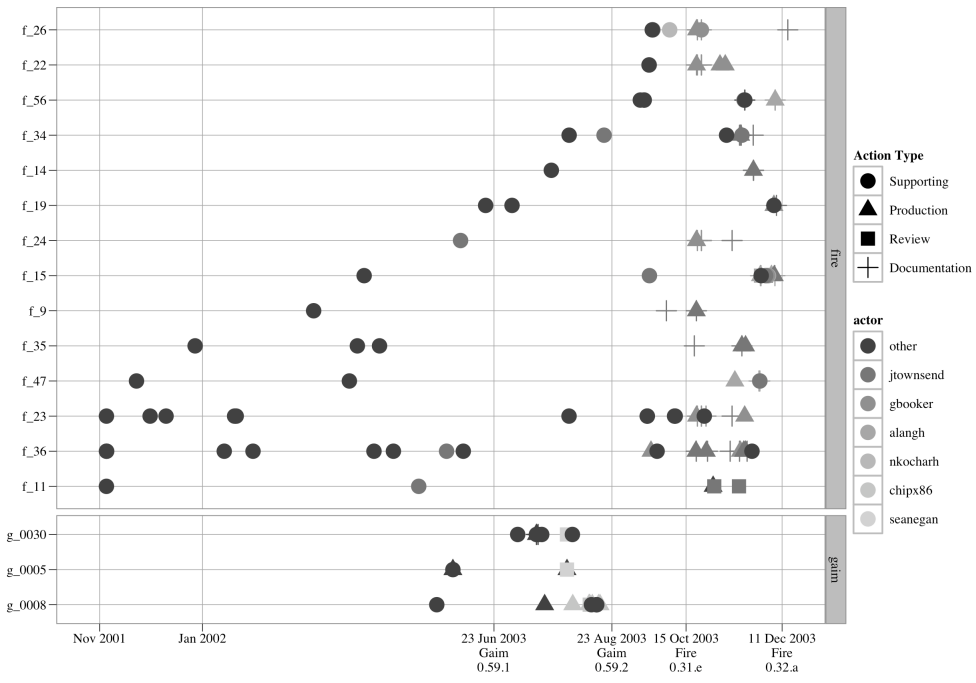

Figure 3. Complex tasks were deferred. The figure shows long running Tasks begun by requests by peripheral participants (black circles). These were accepted by core participants (grey circles) but deferred until the inter-release period, when production work (grey triangles) was undertaken in short tasks with only single programmers.

There was also evidence that complex work was deferred. Figure 2 shows long running Tasks. The early Actions were all Support actions, usually feature requests. Further qualitative investigation of these tasks provides evidence that similar processes of deferral were undertaken; a feature request was acknowledged as desirable, but the work considered too complex to undertake. For example f_9 in Figure 2 consists of a feature request made in March 2003, then assigned to a developer as a signal of acceptance. Yet no work is done until October 2003, when the developer comments that an unrelated feature has simplified the request, "*This is possible now with the 'once' option probably I will check it in the next week or so*". In the specific case of these Instant Messaging clients the addition of protocol specific libraries, written outside the project, facilitated 'waves' of resolution of feature requests. There was no evidence of detailed planning, assignment or breakdown of work towards these tasks, rather the arrival of a new library prompted individual short integrative tasks.

Overall the findings from the participant observation were replicated: the work in Fire and Gaim was undertaken overwhelmingly through individual, short episodes of work. There was very little evidence of planning work (only found in a single task) and no evidence of resource management. Complex work was deferred, rather than being broken down into its components and undertaken collaboratively, only likely to be undertaken when and if a single participant could accomplish it in a short period of time.

Formalization: Theory Elaboration

The replicated findings of the empirical work above pose a number of theoretically interesting questions: how are projects able to build complex interdependent shared artifacts through individual work and how are projects able to progress quickly enough to satisfy their users and developers, given the practice of deferring complex work? In order to answer these questions, and elaborate the growing understanding that the interaction between participant's motivations and the organization of work is a key novelty in FLOSS production, I developed a formal model of developer decisions.

Software, following common industry practice, is modeled as a "stack" of layers, as in Figure 4 and 5.

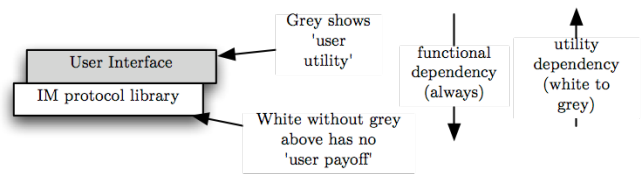

Figure 4. All layers have functional dependencies on those below them. Gray layers have direct utility, and white layers have utility dependencies on gray levels above them.

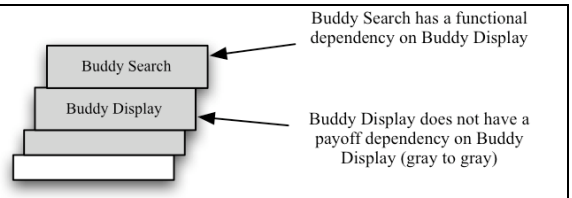

Figure 5. Layers with direct utility can also be built upon, so that there is a functional dependency without a utility dependency, shown with gray on gray.

Table 1: Assumptions (Checkmarks show assumptions relaxed in the dissertation)	
1. Participants will only work for a utility payoff	
2. Participants are motivated only by their own use of the software	✓
3. Participants are good, but not perfect, judges of task complexity	
4. Participants know the limitations of their judgment (expectations approach reality)	
5. All participants have the same set of skills and availability	✓
6. Participants only know their free time for the next turn	✓
7. There are no exogenous sources of code or solutions	✓
8. Participants can build on existing layers without assistance from authors	✓
9. Contributions are always shared under an open source license	

The model begins with a set of assumptions designed to represent a stylized version of the FLOSS situation for analytical clarity, shown in Table 1. Many of these assumptions are formally relaxed in the dissertation; this short paper concentrates on the central model, briefly discussing extensions. Agents are understood to be motivated by

annoyances or limitations encountered in their daily use of the software, conceiving a change to the software code they would like to make. Their decision making is understood to be determined as in the expectancy-valance model of motivation, where "attractiveness of a particular task and the energy invested in it will depend a great deal on the extent to which the employee believes its accomplishment will lead to valued outcomes (Steers, Mowday, & Shapiro, 2004). There are two separate expectancies in this theory, as depicted in Figure 6.

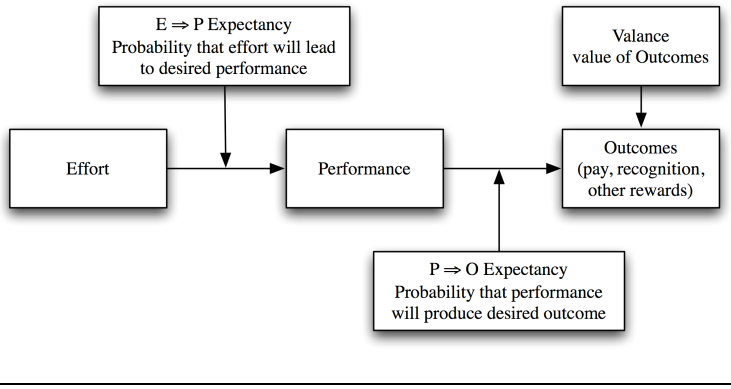

Figure 6. Expectancy-Valance model of motivation. Adapted from (Samson and Daft 2005, p. 534)

The decision equation for the agent, then, is shown in Eq 1.

$$\textbf{(Eq. 1)}\ E(B_{choice}) > E(U_{outcome}) \times E(P(e \rightarrow p)) \times E(P(p \rightarrow o))$$

The agent considers the utility they would derive from a desired change, and considers the current codebase to assess the probability that, given their skills and free time, they would be able to successfully complete the task E(P(e→p))). They also assess whether, having completed their task, they would be able to derive the motivating utility and finally whether the decision to work offers the best use of their free time. Unsatisfied motivations—where the developer either chose not to act or did not successfully complete the work—survive through the next turn (i.e. participants return to the codebase from time to time and reconsider work they previously rejected).

Model analysis

Individual, layered work can proceed. For the case of individual work E(P(p→o)) is set to, and is expected to be, 1 (i.e. certain). This reflects the fact that the agent has the codebase, a compiler and can compile their layer with the application. The simplest situation, therefore, is that a participant inspects the codebase and their set of desires and finds a task which is sufficiently motivating, but is also within their abilities given the time known to be available to them. Such a situation is shown in Figure 7, where a gray layer (depicting utility to its developer) is layered atop an existing gray layer. These situations can be called "backwards-only" dependencies: all that is relied on is the current codebase and its permanent availability, as guaranteed by the FLOSS licenses.

Figure 7. As development proceeds over time through layering, functional dependencies become "backwards only" dependencies. They depend only on what is already present.

The "missing step" problem. Clearly not all work is that simple. An agent will often encounter the situation where their inspection of the current code base would lead them to see that the work needed to implement a desired feature is greater than their skills will allow them to accomplish in the time they know they have available; if they start they will fail. This can be understood as setting E($P(e \rightarrow p)$) so low that alternatives will always provide a higher expected return, regardless of the size of $U_{outcome}$. In a graphic simplification this can be represented by depicting the feature as two separate layers, as in Figure 8.

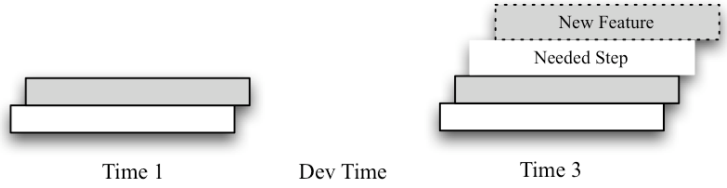

Figure 8. A missing functional dependency becomes both a forwards and backwards dependency; forwards because the New Feature depends on work not yet done.

An individual agent under the assumptions above will not work to try to implement both layers, nor is it rational to work on either in isolation. This is because without the white missing step, the payoff of the gray layer is not available (a missing functional dependency). Without the potential to finish a gray layer in the time available the white layer will not be built (since it is missing a utility dependency). This is a fundamental dilemma in collaboration; it is a stylized and contextualized way of restating a core problem of collective action: the complexity of work is beyond the capabilities of individual participants. The following section presents two solutions to this situation: interdependent collaboration, which is well known, and productive deferral, which is believed to be novel.

Interdependent collaboration. A natural way to resolve this situation is to have agents communicate and potentially make agreements between themselves. In this way they are free to discover other participants who are also motivated by the New Feature. Together they can assess that if both of them work they can accomplish the work; one works on Needed Step and the other on New Feature, as depicted in Figure 9. The capacity to work together opens the way to resolving the dilemma.

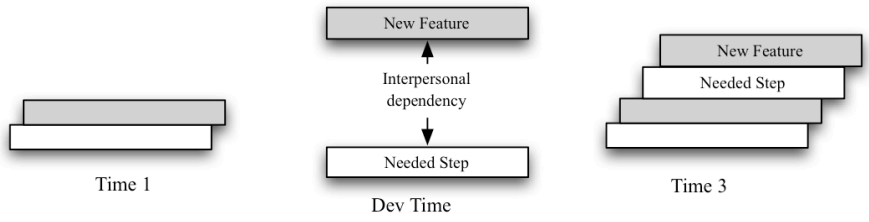

Figure 9. The missing functional dependency can be added if two developers are able to work concurrently, although the risk of non-completion rises due to partner failure and coordination costs.

However such collaboration introduces new risks that could undermine motivation and reduce volunteers' participation in projects. From the perspective of either developer, this agreement introduces a new source of non-completion risk. This is because any developer's collaborator also has a chance of failure, $P(e \rightarrow p) < 1$, and therefore may not complete their part of the agreement, rendering the collective payoff unavailable (either because a functional or a payoff dependency is unsatisfied.) From the perspective of the first developer this means that regardless of their ability to complete their part of the task, the reward may not be available. This can be incorporated into the model as a change in the expectation of the second part of the risk term, $E(P(p \rightarrow o))$, where there is a risk that the expected outcome will not eventuate, regardless of the success in initial performance. Of course this cuts both ways, reducing the expected value of the agreement and effort multiplicatively, sharply decreasing the likelihood of agreement and action. Essentially each agent's $E(P(e \rightarrow p))$ becomes their partner's $E(P(p \rightarrow o))$. For example, if there is an individual chance of success for each developer of 0.8, the overall chance of success is 0.64, meaning that the overall utility of the work would need to be substantially higher if collaboration is

to be a rational choice. Furthermore, concurrent development introduces coordination costs, since participants aren't interacting only with an finished, unchanging codebase, and these costs add multiplicative risk, represented by θ in Eq 2.

$$\textbf{(Eq. 2)}\ E(B_{choice}) > E(U_{outcome}) \times E(P(e{\rightarrow}p)_{self} \times \theta) \times E(P(e{\rightarrow}p)_{other} \times \theta)$$

Since both agents' P(e→p) is less than one and the adjustment for risk derived from coordination costs (θ) is also less than one, work conducted in this manner will always be more risky than if it could have been conducted through individual steps with immediate utility payoffs. What is true for two participants is exponentially worse for three or more participants. When participants experience failures due to interdependence they are less likely to attempt interdependent work again (Kiggundu, 1981). In a volunteer environment the unreliability of participants, and the collective agreement on the priority of "real life", mitigates against collaborative work, shaping the organization of work towards individual tasks.

Deferring complex work. There is another possibility available to developers facing a "missing step" situation, they can defer the work for the current turn (freeing them to pursue other valuable activities), and to wait and see how the codebase changes over time. It is possible that eventually—through the work of others on the project—the situation will have changed sufficiently that New Feature is now possible with only individual work within one period. In short, from the perspective of a single participant, Needed Step simply appears, and turns the dilemma into a relatively simple backwards-only dependency.

What trickery is this? Needed Step is a layer without a utility payoff and, under our assumptions, won't be built through individual work. It can't come from an exogenous source, because we are assuming that these do not exist. How then does Needed Step emerge? Introspection on the participant observation by the author revealed that cognition of developers—their estimation of the work needed to build New Feature—is highly situated: it responds to the codebase as it is now, not to all possible configurations of code. This is modeled by having the developer make their best assessment of the easiest way to change the code to achieve their desired outcomes and bases their calculation of E(P(e→p) on that assessment. The Needed Step/New Feature dilemma is therefore not necessarily a hard fact—it is not a "structural requirement" of a task (Wageman, 1995)—but the result of a developer's cognition. As the codebase changes, new and often surprising ways to accomplish tasks emerge (as illustrated by the emails in Figure 1). Figure 10 depicts this, showing potentially problematic interpersonal dependencies, requiring trust and communication, converted to two backwards-only dependencies, and accomplished through individual work alone.

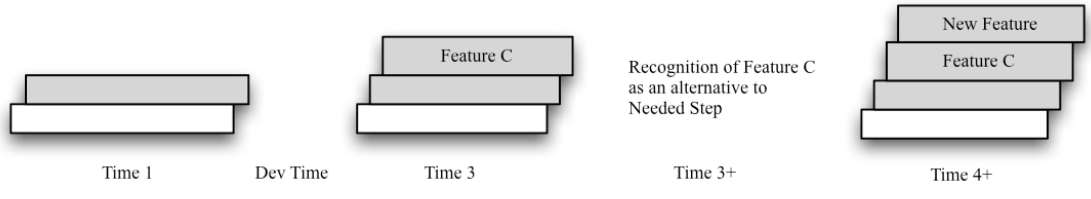

Figure 10. The missing step dilemma (shown in Figure 8) can become two separate backwards-only dependencies, if the missing functional dependency proves to have an alternative which itself has a utility payoff.

Model extensions. There is not sufficient space here to present the full set of extensions undertaken in the dissertation. For example, the model is extended to account for sequential work agreements (where coordination costs are reduced, but a discount factor accounts for the delayed gratification). The model is also extended to consider a non-uniform distribution of skills, finding that this increases the chance that others will implement "missing steps" because they find them easier, suggesting that the common idea of FLOSS "code gods" reflects this.

Most importantly the model is relaxed to incorporate experiential motivations, such as learning, which don't depend on successful completion of the task to be useful; they are an additive term in the decision equation. The dissertation argues that these motivations are especially useful to projects, allowing "needed steps" to be built as well as encouraging participants to "have a go" even when the challenge seems beyond them, and sometimes succeeding. The model also clarifies formally why exogenous sources of code, such as libraries, are so important as they are 'bundles' of features, which once integrated allow others to exploit their full potential, speeding the project through "missing step" dilemmas.

Discussion

The theory developed in the dissertation emphasizes how much can be accomplished without risky interdependent collaboration (particularly when intrinsic motivations, such as learning, are considered). The theory is useful in part because it focuses attention on two aspects vital to efforts to adapt the practices of FLOSS collaboration in other areas.

Firstly the model draws attention to the importance of the IT artifact at the center of these collaborations (Benbasat and Zmud 2003; Orlikowski and Iacono 2001). I do not mean the IT used to communicate or host the software, but the software itself. Software as an object of collaboration, and particularly distributed collaboration, has affordances that are key.

Firstly software can be built in relatively small layers, particularly if an appropriate architecture is chosen. Those small additions can be relatively easy to integrate and can have independent payoffs, even for very small additions ("stackable incentives"). Much work is not like this. For example long-form communicative work, such as in reports or novels, requires attention effort from the receiver and therefore it is vital to deliver them as parsimoniously as possible. Similarly, an airplane certainly can be built in layers: first the fuselage, then the engines, and finally the wings. But until it is complete none of these steps provide any utility payoff.

Secondly, software has ultra-low instantiation and distribution costs. By instantiation costs I mean the costs of moving from a design to a useful artifact. For software this is cheap, simply a computer and a compiler, but for other work, such as building a house, this can be very expensive. Ultra-low distribution costs are also very important if the small layers are to spread out to the community and provide the basis for other's work. Prior to Internet software delivery the instantiation and distribution costs of updates involved printing CDs and shipping them to customers, a much more expensive proposition.

Finally software is rewindable; as participants add layers they do not commit the entire project to retaining them forever; changes can be undone, especially when source code management systems like CVS are used. This is quite unlike the great majority of other work, such as customer service or financial advice, where actions and their impact are hard or impossible to reverse. Conversely, Wikipedia and the artifact at its center, seem to share the features of software and should, therefore, also support layered collaboration.

The second area of focus for adaptability is on the feasibility of deferral as a productive mechanism in traditional IS development. Here, of course, the context of software development is quite different. In broad strokes, the payoff and motivation for the work is sales of software to customers or internal use of software to support a business, rather than its intrinsic value or the enjoyment of the experience. This necessitates paying the developers. Since this has to occur prior to the sale or internal use, investment is required. One consequence of investment is the operation of the time cost of money, meaning that any delay in releasing the software is doubly costly; not only must payroll be maintained but the opportunity cost of the investment capital begins to mount. This is compounded when competitive pressures, such as first mover advantage, are in play. All this is a long way of saying that businesses face deadlines and deferral is unlikely to be a valued software development strategy. When a business is relying on open mechanisms but faces a deadline they may attempt to "speed-up" development through the application of paid developer's time. Yet this must be done judiciously, since it risks undermining the motivation of volunteer participants.

Together these considerations raise serious concerns about the adaptability of the FLOSS organizational model in other environments, especially in businesses.

Conclusion and Contribution

This dissertation makes useful and significant contributions, albeit not without limitations. The primary limitation of the dissertation is the decision to trade empirical generalizability beyond three specific FLOSS cases for depth needed for theory development. Nonetheless the work is the first to draw together the motivations of participants, the technologies of collaboration and the experience and organization of production into a novel theory with practical implications for research and practice in the fields of Information Systems and Organization Science.

The work makes a contribution to Information Systems because it is a socio-technical theory where the IT artifact plays a double role, both as the object of collaboration, as discussed in the Discussion above and as the medium of

collaboration, as the leanness of communication technologies mitigates against potentially problematic interpersonal reliance in a volunteer environment. The theory has implications for the adaptability of FLOSS methods to traditional IS development and for the interaction of the IS function in organizations with FLOSS communities. A contribution is also made to Organizational Science because a new theory of organizing is described and analyzed, where the task is not structurally determining the appropriate way to organize, but rather appropriate organization is emergent, shaped by the volunteer resource context and flexible technologies of production and collaboration. Finally the concept of "strategic deferral" is believed to be novel and may find application in other organizational domains, such as entrepreneurial entry strategy.

The organization of community-based FLOSS development seems to provide a way forward for work outside the boundaries of traditional invest-and-control organization, offering autonomous, satisfying and useful work. Working together alone—individually in company—may offer a way forward, a way out of the traps of exploitation and alienation common across many fields of human endeavor. Yet the work in this dissertation sounds a strong note of caution in that assessment. The components of the socio-technical configuration described here reinforce each other, creating a bundle which works in its place, but which will likely prove tricky to unpick for piecemeal adaptation in other kinds of work. Yet as far as this can be done—and it is far from easy—FLOSS may indeed realize its promise of a truly optimistic and emancipatory future of work.

References

Barnard, C. I., Andrews, K. R., & Andrews, K. R. (1968). *The Functions of the Executive*.

Cook, S. (2008). The Contribution Revolution: Letting Volunteers Build Your Business. *Harvard Business Review*.

Crowston, K., Wei, K., Li, Q., Eseryel, U. Y., & Howison, J. (2005). Coordination of Free/Libre Open source software development. In *ICIS 2005. Proceedings of International Conference on Information Systems 2005*. Las Vegas, NV.

Drucker, P. (2002). They're not Employees, They're People. *Harvard Business Review*, *80*(2), 70–77.

Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K. (2005). *Perspectives on Free and Open Source Software*. Cambridge, MA: MIT Press.

Ghosh, R. A., Robles, G., & Glott, R. (2002). *Free/Libre and Open Source Software: Survey and Study FLOSS*. University of Maastricht: Netherlands: International Institute of Infonomics. Retrieved from http://wwww.infonomics.nl/FLOSS/report/.

Handy, C. (1988). *Understanding voluntary organizations*. London: Penguin Books.

Herbsleb, J. D., Mockus, A., Finholt, T. A., & Grinter, R. E. (2001). An empirical study of global software development: Distance and speed. In *the International Conference on Software Engineering (ICSE 2001)* (pp. 81–90). Toronto, Canada.

von Hippel, E., & von Krogh, G. (2003). Open Source Software and the `Private-Collective' Innovation Model: Issues for Organization Science. *Organization Science*, *14*(2), 209–223.

Kiggundu, M. N. (1981). Task interdependence and the theory of job design. *Academy of Management Review*, *6*, 499-508.

Lakhani, K., & Wolf, R. G. (2003). *Why hackers do what they do: Understanding motivation efforts in Free/F/OSS projects*. Working Paper, MIT Sloan School of Management.

Lakhani, K., & von Hippel, E. (2003). How open source software works: "free" user-to-user assistance. *Research Policy*, *32*, 923–943.

Lerner, J., & Tirole, J. (2002). Some simple economics of Open Source. *Journal of Industrial Economics*, *52*(2), 197–234.

Malone, T. W. (2004). *The Future of Work: How the New Order of Business Will Shape Your Organization, Your Management Style and Your Life*. MIT Press.

Malone, T., & Crowston, K. (1994). The interdisciplinary theory of coordination. *ACM Computing Surveys*, *26*(1),

87–119.

McLure Wasko, M., & Faraj, S. (2005). Why Should I Share? Examining Social Capital and Knowledge Contribution in Electronic Networks of Practice. *MIS Quarterly*, *29*(1), p35 - 57.

Samson, D., & Daft, R. (2005). *Management - Pacific Rim Second Edition*. Sydney, Australia: Thomson.

Shea, G. P., & Guzzo, R. A. (1987). Group effectiveness: what really matters? *Sloan Management Review*, *28*(3), 25-31.

Steers, R. M., Mowday, R. T., & Shapiro, D. L. (2004). The Future of Work Motivation Theory. *Academy of Management Review*, *29*(3), 379–387.

Thompson, J. D. (1967). *Organizations in Action: Social Science Bases of Administrative Theory*. New York: McGraw-Hill.

Van de Ven, A. H., Delbecq, A. L., & Koenig, R. (1976). Determinants of Coordination Modes Within Organizations. *American Sociological Review*, *41*(2), 332–338.

Wageman, R. (1995). Interdependence and group effectiveness. *Administrative Science Quarterly*, *40*(1), 145-180.

Wageman, R., & Gordon, F. M. (2005). As the Twig Is Bent: How Group Values Shape Emergent Task Interdependence in Groups. *Organization Science*, *16*(6), 687–700.

Yamauchi, Y., Shinohara, T., & Ishida, T. (2000). Collaboration with Lean Media: How Open-Source Software Succeeds. In *Proceedings of Computer Support Collaborative Work 2000 (CSCW 2000)*.