MIS Quarterly

RESEARCH ARTICLE

COLLABORATION THROUGH OPEN SUPERPOSITION: A THEORY OF THE OPEN SOURCE WAY[1]

James Howison
School of Information, University of Texas at Austin, 1616 Guadalupe Avenue,
Austin, TX 78701 U.S.A. {jhowison@ischool.utexas.edu}

Kevin Crowston
School of Information Studies, Syracuse University, 343 Hinds Hall, Syracuse, NY 13244 U.S.A.
and National Science Foundation {crowston@syr.edu}

This paper develops and illustrates the theory of collaboration through open superposition: the process of depositing motivationally independent layers of work on top of each other over time. The theory is developed in a study of community-based free and open source software (FLOSS) development, through a research arc of discovery (participant observation), replication (two archival case studies), and theorization. The theory explains two key findings: (1) the overwhelming majority of work is accomplished with only a single programmer working on any one task, and (2) tasks that appear too large for any one individual are more likely to be deferred until they are easier rather than being undertaken through structured team work. Moreover, the theory explains how working through open superposition can lead to the discovery of a work breakdown that results in complex, functionally interdependent, work being accomplished without crippling search costs. We identify a set of socio-technical contingencies under which collaboration through open superposition is likely to be effective, including characteristics of artifacts made from information as the objects being worked on. We demonstrate the usefulness of the theory by using it to analyze difficulties in learning from FLOSS in other domains of work and in the IS function of for-profit organizations.

Keywords: Open source, information systems development, materiality, socio-technical system, collaboration, coordination

Introduction

The success of new ways of organizing closely associated with information systems, such as open source software and Wikipedia, is surprising because these modes of work blend three circumstances previously found to be challenging: working at a distance (e.g., Lipnack and Stamps 1997; Olson and Olson 2000), working with sporadically available volunteers (e.g., Dunlop 1990; Handy 1988) and working on complex artifacts such as software and documents (e.g., Herbsleb et al. 2001; Kittur and Kraut 2008). Unsurprisingly, then, many researchers and managers look to these ways of organizing for inspiration for virtual work generally, hoping to learn from their example (e.g., Agerfalk and Fitzgerald 2008; von Krogh and von Hippel 2003; Scacchi et al. 2006; Stewart and Gosain 2006). Accordingly there is a need to develop theory that provides insight into how such ways of organizing function and under what conditions they are likely to be successful and transferable to other virtual collaboration settings.

Key to our empirically based theory are the material characteristics of the work done in these settings, specifically soft-

[1]Suprateek Sarker was the accepting senior editor for this paper. Brian Butler served as the associate editor.

The appendices for this paper are located in the "Online Supplements" section of the *MIS Quarterly*'s website (http://www.misq.org).

ware as an *object* of collaboration—that which is worked on. The material characteristics of the object being produced affect how one can successfully organize around its production because they "provide capabilities that afford or constrain action" (Leonardi 2010). The theory in this paper is a theory about the work of building artifacts out of information and the organizational affordances of these artifacts (see Zammuto et al. 2007).

Prior research on virtual collaboration has not focused on the specifics of the task undertaken. Instead it has emphasized the ways virtual teams engage in work that crosses space and time (Sarker and Sahay 2004). Such crossing is accomplished by using technologies as a *medium* of collaboration, a mechanism of bridging, drawing together participants across a set of discontinuities including time, space, organizational, and cultural boundaries (Watson-Manheim et al. 2002). Research on virtual collaboration has examined collaborative processes such as leadership (Yoo and Alavi 2004), bringing different cultures together (Jarvenpaa and Leidner 1999) and achieving virtual presence to manage conflict (Hinds and Mortensen 2005). Even research on task–technology fit has been concerned with characteristics of communication media, such as richness and synchrony, and fit with abstract aspects of task, such as certainty (Dennis et al. 2008), rather than specific characteristics of the artifacts being built.

The situation is similar in cognate disciplines, including small group research, which have focused on what Rousseau et al. (2006) call *team work*, (such as leadership, communication, decision making, adaptability), setting aside *task work* as overly specific. The focus away from specifics of work has also occurred in organizational science and even in industrial engineering (Bailey and Barley 2005; Barley and Kunda 2001).

It is ironic that the focus of virtual teams research in Information Systems has not been on specific characteristics of the task undertaken because the development of software has long been a clear focus of IS research (e.g., Ives et al. 1980). Yet when IS researchers study the work of building software in virtual teams, it is primarily as an example of a more generalized concept of work (e.g., Sarker and Sahay 2004), rendering the technologies worked upon indistinct, technology as "work in progress," as Orlikowski and Iacono (2001, p. 126) term it. This lack of attention to the task work itself persists even as work throughout the economy increasingly involves manipulating information, such as design, customer service, editing knowledge databases, and building websites, prompting calls for theorizing the sociomaterial or sociotechnical and its relationship to organizing (Bailey and Barley 2005; Leonardi and Barley 2008; Orlikowski and Scott 2008; Zammuto et al. 2007; Zuboff 1988). Research focusing on the objects of work and their relationship to organizing is rare (Zammuto et al. 2007), with Malhotra and Majchrzak's (2012) study of the ability to annotate documents in a shared repository and its role in team situational knowledge evolution a recent, rare example.

We examine work on free (libre) and open source software (FLOSS). FLOSS development is a canonical type of distributed, online production and is seen as an important source of potential lessons for virtual collaboration throughout the social sciences, from Information Systems (e.g., Agerfalk and Fitzgerald 2008; Crowston and Wade 2010; Stewart and Gosain 2006), to Organization Science (e.g., von Krogh and von Hippel 2006) and beyond (e.g., Duval 2010; Shirky 2008). In particular "the open source way" is interesting because it is seen as different from existing high-level models of collaboration, such as hierarchy, markets, or networks (Benkler 2002; Powell 1990). The projects studied in this paper are community-based FLOSS projects, with no formal institutional existence (such as a nonprofit foundation like the Apache Software Foundation) and no significant corporate involvement (unlike Apache or Linux). The projects chosen epitomize what is most novel in the FLOSS phenomenon, its least hybridized form. Thus they provide appropriate grounds for theorizing about what is different about FLOSS as a model for collaboration and organizing.

Research on FLOSS development has also tended to focus on team work rather than task work, identifying factors associated with success that most plausibly generalize to other settings. These include control (Gallivan 2001), governance (O'Mahony and Ferraro 2007), ideology (Stewart and Gosain 2006), past collaborative ties (Hahn et al. 2008), and knowledge flow between projects (Daniel and Diamant 2008).

In contrast, less research focuses on FLOSS development task work and links it to the wider context (e.g., Krishnamurthy 2002; Yamauchi et al. 2000). Academics with a strong practitioner background in FLOSS have emphasized understanding the volunteer environment as fundamental to the organization of successful FLOSS task work (Capiluppi and Michlmayr 2007; Michlmayr 2004). They have linked volunteering to the architecture of the software, arguing that more modular structures will attract more volunteers (Baldwin and Clark 2006; Conley and Sproull 2009; MacCormack et al. 2006) and give the projects *actionable transparency*, which allows volunteers to quickly and usefully engage (Colfer and Baldwin 2010). Research drawing on the job design tradition has focused on the motivational impact of "job-related" rather than "person-related" characteristics (Hertel 2007), finding that the most motivating tasks are those with three characteristics: they satisfy a need for competence, they provide

high autonomy, yet they also preserve relatedness between participants (Ke and Zhang 2010). Our work advances this line of inquiry into the importance of the specific tasks undertaken in FLOSS development.

We argue that *collaboration through open superposition* is at the core of the success of community-based FLOSS projects, allowing an organization of task work that leads to the discovery of a work-breakdown that is both motivating and surprisingly coordinated. This is accomplished while minimizing the types of interdependency that team work processes are required to manage. Our theory allows us to identify a particular set of conditions for the success and potential adaptation of this way of working, thus helping those seeking to learn from what is popularly called "the open source way" and benefit from the potential of outsourcing to an unknown workforce (Agerfalk and Fitzgerald 2008).

This paper presents empirically grounded and illustrated theory development (Weick 1989, 1995), undertaken though an unfolding arc of *discovery*, *replication*, and *theorization*. *Discovery* reports on four years of participant observation in a community-based FLOSS project, *replication* describes an archive-based field study in two similar community-based FLOSS projects. Finally *theorization* explains (1) the reason for the observed patterns of work, (2) how such patterns are successful in building complex software, and (3) the contingencies under which such patterns are likely to be successful. In the discussion, we demonstrate the usefulness of our theorizing by examining the challenges of adapting community-based FLOSS organization for other types of work and institutional environments, especially the IS function in firms. We include a comprehensive methodological appendix (Appendix C) to provide additional details on our overall method of theory development and the specific methods of each study.

Discovery: Participant Observation

For over four years, the first author participated in and observed the BibDesk project (http://bibdesk.sourceforge.net/), a community-based FLOSS project producing a reference manager akin to EndNote. The first author was prepared for this through studying case and ethnographic methods in Information Systems (Harvey and Myers 1995; Myers 1999; Yin 1994) and by reading relevant ethnographies (Barley 1986; Knorr-Cetina 1999; Yates and Orlikowski 1992; Zuboff 1988).

Participant observation derives insight from reflection on embedded, longitudinal, lived experience (Lewis 1985; Myers 1999). The first author, therefore, specifically chose a product that could be integrated into his lived experience as a graduate student (a bibliographic manager). This project spanned four years in field (exceeding standard recommendations for ethnographic length; e.g., Yin, 1994, pp. 10-11), cycling up and down for active work episodes, but continually maintaining daily contact with the project through use of the application, as well as subscriptions to mailing lists and bug trackers, in the manner of open source "natives."

Throughout this period, the first author maintained field notes and produced periodic thematic memos for discussion, as recommended by Myers (1999), with the key principle that memos capture understandings that were surprising, before they become commonplace. These notes were frequently discussed with the second author (Walsham and Sahay 1999) and a wider research group. The evolving understandings were "disciplined" toward plausibility and interestingness (Weick 1989) through interaction with both the evolving academic discourse on FLOSS and in interactions with FLOSS practitioners through presentations and discussions at conferences such as O'Reilly OSCON and ApacheCon. We present additional details on the methods of participant observation and analysis, including examples, in the methodological appendix.[2]

Into the Field

I let my case emerge naturally from my day-to-day practice as an academic, adopting FLOSS tools wherever possible. BibDesk supported my day-to-day writing work well. The BibDesk project has always been open source. It was founded by a graduate student at the University of California in San Diego and many of the participants were fellow graduate students. All were volunteers and none met face to face, making BibDesk a good non-hybridized case of community open source. Within its niche, BibDesk is a successful project: it has consistently been in the very top percentile of active Sourceforge projects and, as of October 2008, listed 13 developers, although only 5 were consistently active during my observation period. While much attention has been paid to very large FLOSS projects, such as the Linux kernel, the majority of FLOSS projects have profiles similar to BibDesk (Crowston et al. 2012).

The working life of BibDesk occurs in a number of different communication venues and the deeper one's participation, the

[2]The following sections are written in the first person from the perspective of the first author, highlighting the epistemological origins of the understandings it presents.

more venues one encounters. My first encounter with the project was through the application itself; I found the shared experience of using the application to be an important part of being involved in the project. The project mailing lists, bibdesk-users and bibdesk-dev, were the next venues I encountered. My first message to the list suggested a feature improvement; in reply the founder gently and encouragingly directed me to a specific section of the code.

Fired up, I downloaded ("checked out") the code from Source Forge and attempted to build the project from source. This introduced me to another project venue, the source code repository, in the case of BibDesk, a system called CVS (Concurrent Versions System). However, when I first attempted to compile the application, it did not successfully build. This outcome was a frustrating experience that immediately undermined my motivation to contribute. Girded, however, by commitment to the project as a research setting, I determined that the source of the errors was related to the location of external libraries (which, by default, was different from the location that the developers had changed manually on their machines).

My first contribution, therefore, was a Perl script to download these libraries to the correct directory, then check out the BibDesk source and build the application. I submitted this script to the developer mailing list where it was well received: the developers hadn't known that their source didn't build easily on the machines of potential contributors. This task was also my first introduction to the project's unit of contribution: the patch. A patch is a set of changes, a "diff" that is applied to the codebase adding new functionality or fixing bugs. As developers work on the code, they share that work with other developers by submitting patches. These patches can be sent to the mailing list, but are more commonly used to update the shared copy of the source in CVS, which is called "making a commit." The resulting process is somewhat like coauthors sharing Word documents with tracked changes (with far more control and history).

With longer participation, I also came to encounter another important venue: trackers. Trackers are issue management systems, resembling web forums. BibDesk used just the two default types of trackers provided by Sourceforge: Bugs and Request for Enhancements (new features). While most discussion remained on the mailing list, trackers were used as longer-term memory, especially useful when mailing list threads became fractured by long pauses in a discussion.

As I became more involved in the project I found that my understanding of the life of the project was not organized by any technological feature of any of these tools, such as threads or tracker items, but by episodes of work in which the developers and users were engaged, which we call tasks. Tasks provide coherence to work but leave traces scattered throughout different venues. Thus a task might begin with messages on a mailing list, continue through posts in a tracker, then involve a patch, a CVS check-in, and finally a functional change to the application itself. Others tasks might simply show up in CVS, then in the application.

Three Vignettes of Work in the BibDesk Project

Container Column

A task I undertook as part of the BibDesk project was to create a new kind of column in the summary display labeled "Container." This column displays the journal title for articles and conference or proceedings title for conference proceedings (journal and conference title are separate fields in a BibTeX record and so would otherwise be displayed in two separate columns). I undertook this task in April 2005, motivated by personal annoyance at the small screen of my laptop, which made it difficult to see both columns at once.

I worked on this task in private, without sharing my plans with the project beforehand. I did this because I thought I had a good understanding of what I wanted to achieve and did not want to bother the other developers with simple questions about the code, especially if I wasn't able to complete the task. My first public discussion of the idea was an e-mail to the developer's list, describing the feature and including a patch. I hadn't committed it to CVS—even though I had commit privileges—because the patch didn't work quite as I'd hoped (it wouldn't sort properly), but worked well enough to show my intentions. The project founder reviewed the patch and replied, endorsing the intended change and providing comments on how to fix the sorting issue. I found this motivating (especially as the fix corrected an embarrassingly simple error I had made) and so after 4 hours further work, I fixed the sorting and committed the patch, thus making the results of my work available to other developers and users. In total, I estimate that it took about 20 hours of work spread over 3 days.

This episode was fairly typical of my involvement, and, by observation (and an archival reconstruction not reported here), the patterns of involvement of other developers. Tasks tended to be primarily undertaken by an individual programmer in a relatively short period of time at the developer's own behest, motivation, and timing. Support between developers, if there was any, was unplanned—more a case of reaching out on the

chance that someone was there than a case of planned interdependency. Tasks resulted in a single patch that bundled up the changes necessary to make incremental and immediately useful progress.

BibDesk 2.0

The second episode illustrates a markedly different type of task. The BibDesk 2.0 episode was a long running period in which the intention of the group was to release a re-factored and largely rewritten BibDesk. Yet as of December 2012, the current version was 1.5.10, which is to say that BibDesk 2.0 never emerged. The work on version 2.0 began in the same way as BibDesk itself: as a private project of the project founder that eventually moved into BibDesk's public repository. One of the main intentions was to satisfy a persistent user request to move BibDesk from its roots as a BibTeX-centered project to a generic reference manager able to be integrated with document preparation systems other than LaTeX. BibTeX was to be replaced as the underlying file format and instead be simply one export format among many. In addition, it was the stated intention that the work would make contribution easier by re-factoring the codebase.

The 2.0 project, however, never caught hold. I managed to build it a number of times, but the functionality was low compared to the 1.0 version and none of the developers could adopt it for day-to-day work. Rather than switching work to the BibDesk 2.0 version, the participants (other than the project founder) largely continued to tweak BibDesk 1.0. There was no significant tension about this situation, but the reality was that it was hard for the rest of the project to change tack and focus on BibDesk 2.0, even though the participants generally agreed with a need for a rewrite and did contribute some small testing and work on BibDesk 2.0 (which remains in the code repository). Instead work continued on BibDesk 1.0, making progress in small steps.

Even without shifting focus, the developers eventually achieved most of the features planned for 2.0: a vastly improved group system, a very flexible non-BibTeX template system, the ability to store more than one file per entry and to use file aliases instead of full paths. However, the project achieved this outcome through small additions over time, retaining the basic architecture of the 1.0 software and even the reliance on the BibTeX file format.

The BibDesk 2.0 episode was problematic because it envisaged a radical reconfiguration of the relationships between the developers. Specifically, working on version 2.0 left other developers dependent on the completion of work by the project founder, or, had others joined the effort, interdependent on contributions by each other, where the payoff in working software was weeks if not months down the track, a mode of work quite different than the normal pattern of incremental small steps with immediate payoffs. Even though the group took a consensus decision to hold off adding new features to the 1.0 codebase, as time stretched forward this agreement was tacitly abandoned, the developers returning to their non-interdependent incremental development process.

Web Groups

The third vignette shows that this incremental, layered process is surprisingly capable. The vignette is illustrated by two emails from the project founder, written four years apart. The first e-mail is a response to a suggestion I made regarding a feature to subscribe to publication lists on an academic's personal homepage. The project founder had previously conceived the idea and agreed that it would be a useful feature but never began work on it. In 2003 he wrote,

> *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****).*

I also found the task too complicated for the time I was able to devote to the project. The task was thus left languishing. Four years later, somewhat out of the blue, the project founder checked in a patch that implemented the 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****.*

The founder emphasized that the task had become "much easier" in the intervening years because of the incremental layered work of other developers—work undertaken for other features that just happened to also support Web Groups. The work undertaken while the task was languishing had prepared the ground so that a developer working alone could, in a matter of days, complete a feature that earlier had been too much work to even begin.

Participant Observation Findings

Organization and Interdependency

The unit of contribution in the project was the patch, which wraps up code changes associated with a particular task.

Project members built on each other's work, but just as a patch alters only what is there already, developers very rarely relied on each other's future availability or planned work. They did not work entirely alone, but sought and gave support only spontaneously. Tasks tended to be relatively short, on the order of a few days of work. Work that did not fit this model, such as BibDesk 2.0, was difficult to complete, sometimes failing entirely. At other times, such work was deferred and revisited only when other independent work had—in an unplanned way—changed the codebase so that the work could be accomplished in short independent tasks.

Motivation and Organization

While motivation is an often-studied topic in research on FLOSS, it has only been studied through surveys and interviews. Participant observation affords the addition of introspection. My experience pointed to the involvement of two aspects of volunteer motivation that fit with the organization of work above. First, 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. For me, the goal of working on the project was an application that worked better, not the experience of coding (i.e., the journey was not the destination). Work that would not lead to such an outcome was not very motivating. Second, I worked alone because I did not want to rely on the free time and commitment of others to finish something I would need for my work. In part, I felt I had no right to request their time. More importantly, since their commitment was also unpredictable, I did not want to rely on their portion of shared work being completed. I feared being left "high and dry" if they, quite legitimately, had to attend to their real life instead of the project.

Collaboration Through Superposition

The identification of the patch as the unit of contribution led us to the conceptualization of superposition as vital to the way software is produced in the BibDesk project. Work proceeded in small, independent tasks, each with a functional pay-off through its changes to the codebase and thus the application. These changes layered on top of each other over time, each conceived and implemented for their own sake, yet simultaneously creating the circumstances taken as given for the production of the next layer in a way analogous to the superposition of rock strata.

Superposition through layering is a way of understanding software—and its construction over time—that is related to, but distinct from, modularity. As mentioned above, modular architectures have been suggested as fundamental to good software and to attracting volunteer participants (Baldwin and Clark 2006; Benkler 2002; Conley and Sproull 2009; MacCormack et al. 2006). A module has as its distinguishing characteristic its separateness from other code, as measured by low coupling, and the manner in which it groups related functionality, as measured by high cohesion (Parnas et al. 1981). By contrast, a software layer, as conceived in this paper, may draw on code from many functional modules to deliver its payoff; its distinguishing characteristic is that it takes as its starting point only what is already there. Indeed, most patches in BibDesk seemed to span across formal modules, since they were mostly focused on delivering new functionality rather than optimizing code. Modularity may support the production of software in layers by reducing the amount of the codebase that needs to be altered and thus understood. But they are not the same thing: modularity is a characteristic of the codebase, while the superposition of layers is a characteristic of its production.

Replication: Archival Case Studies

To challenge and strengthen the insights gained from participant observation, we undertook an archive-based field study to see whether the layering of short, individual episodes of work and the deferral of complex work were significant factors in similar cases. If not, then their appearance might simply have been particular to BibDesk; if so, then it would suggest a consistent phenomena worth theorizing about.

Case Selection

Two cases similar to BibDesk and to each other were selected for the replication analysis—Fire and Gaim—both instant-messaging clients. They are similar in that they are entirely volunteer-based and without revenue, corporate involvement, or foundations. They are hosted on Sourceforge and use similar tools. They have roughly comparable numbers of developers: at the time chosen for study each project numbered approximately 10 developers, although 2 years later Gaim had accelerated to having over 25 active developers. As with BibDesk, this places the projects at the higher end of FLOSS development team size. Both projects were relatively successful, Gaim having been Sourceforge Project of the Month. Fire and Gaim are also similar to BibDesk in that they build applications that are personally used by their developers. Finally, the two projects are comparable to each other because they develop similar applications.

Table 1. Inductive Concepts Used in Organizing Archival Records

Concept	Definition — Example
Document	Archived content — An e-mail message, tracker comment, release note
Event	An *event* causes *documents* to be archived — Sending an e-mail, releasing a version
Participant	A distinct individual involved with the project — Larry Wall (the person), Sean Egan (the person)
Identifier	A string identifying a *participant* — Larry Wall (the name), larry@wall.org (an e-mail address)
Task Outcome	A change to the shared output of the project —A new feature, a fixed bug, updating documentation
Action	Work that contributes to a *task outcome* — Writing a translation, requesting a feature, writing code
Task	The sequence of *actions* contributing to a particular *task outcome* — Creating "buddy search"

Data

Participant observation indicated that work proceeds across many project venues, and a coherent understanding of the project's organization could not be obtained from single venues, such as the mailing list. Therefore, data collection was as comprehensive as possible, including mailing lists, source code repositories (CVS and SVN), forums, issue trackers, and release notes. We analyzed an inter-release period (approximately 45 days) for each project, chosen to be close in calendar time so work on the projects would be dealing with similar external contexts.

Analysis

The overall purpose of the archival analysis was narrative reconstruction, focusing particularly on sequence and focal actors (Pentland 1999). The reconstruction of the work episodes from archival records relies on a method that Geiger and Ribes (2011) would later call *trace ethnography*. This involves a process of inversion (Bowker and Star 2000) that connects back from digital trace records to lived experience drawing on an ethnographic understanding of the "socio-technical infrastructure of documentary practices" (Geiger and Ribes 2011, p. 5) that is built during participant observation. As described further in the methodological appendix, the analysis process was disciplined in three ways: grounding task selection in the participant's own records, exploring changes within the well-structured code repository, and seeking narrative cohesion (Pentland 1999).

Starting from the participant observation and working inductively, we developed a set of concepts to describe the work as a set of tasks, shown in Table 1. We defined a *task outcome* as a change to the shared outputs of the project, usually the software but also potentially including documentation or the project website. A *task*, then, was a series of *actions* undertaken by *participants* contributing to the *task outcome*. *Actions* could be directly observed in the participant observation study, but in the archival study we relied on *documents*, such as e-mails, CVS check-ins, and log messages, to provide evidence of these *actions*.

Using this framework, we assembled evidence of the *tasks* performed in each project, in terms of *participants*, *actions*, and *outcomes*, from the evidence in the collection of *documents* for each project. We did this by organizing the archive into collections of *documents* for each *task*.

We began with the Release Notes, and the README file—literally, a file in the source code named README, containing notes from the developers about the code, updated as the code is updated, and including the participants' own description of *task outcomes*. *Documents* from the various data sources relevant to each *task* were then grouped together. However, the Release Notes and changes to the README file do not necessarily record all completed *task outcomes*, so we worked iteratively until we had assigned all of the records from the source code repository (since *tasks* all necessarily alter this shared work product), creating new *tasks* as needed. A total of 106 *tasks* were identified, 62 for Fire and 44 for Gaim, 65 from the Release Notes, 31 from changes to the README file, and 10 from changes to the source code repository alone. Once we had a set of *tasks*, each described by an outcome and including a collection of *documents*, the *documents* were examined to identify the *actions* contributing to the *task outcome*. *Actions* were classified using the inductively developed coding scheme shown in Appendix A (top level codes were management work, review work, production work, documentation work, and support work). *Actions* were also coded for their timing and the *participants* involved. Table 2 shows sample *tasks*, with their *actions* and the codes applied to them. The top cell shows a task in which multiple programmers worked, which is a common image of collaboration. The bottom cell shows contrasting examples of when only a single programmer worked on a task.

Table 2. Illustrative Tasks

Co-Production			
Date/Gap	**Participant (role)**	**Action**	**Code Applied**
Gaim Task 2: Manual Browser Security Fix			
20 July 2002	kareemy (user)	reports bug	Use Info. Provision
1D 5h 50m=	lschiere (dev)	attempts diagnosis	Code Info. Provision
(undated)	robot101 (p dev)	writes patch	Core Production
20D 9h 41m	seanegan (dev)	checks in patch	Review
10D 18h 10m	seanegan (dev)	tweaks fix	Polishing
1D 20h 8m	chipx86	re-writes fix	Core Production
1D 3h 20m	seanegan (dev)	move fix to branch	Management
Solo Production			
Date/Gap	**Actor (role)**	**Action**	**Code Applied**
Fire Task 57: User List Duplicate Fix			
06 Dec 2002	gbooker (dev)	fixes bug	Core Production
Gaim Task 3: Iconv Library Integrated			
02 Aug 2002	seanegan (dev)	adds library	Core Production
19m 52s	seanegan (dev)	edits ChangeLog	Documentation
26m 10s	seanegan (dev)	integrates library	Core Production
Fire Task 5: Scroll on PgUp			
19 Nov 2002	nkocharh (p dev)	makes PgUp scroll	Core Production
Fire Task 29: AIM Buddy Icons			
27 Oct 2002	gbooker (dev)	checks in buddy icon code	Core Production
(same time)	gbooker (dev)	changes ChangeLog	Documentation
39m 3s	gbooker (dev)	add jpg icons	Polishing
1h 22m	gbooker (dev)	add bitmap icons	Polishing
12h 1m	gbooker (dev)	.buddyicon save	Polishing
1h 22m	gbooker (dev)	add bitmap icons	Polishing
3D 13h 1m	gbooker (dev)	fix IRC icons	Polishing
3D 18h 34m	gbooker (dev)	fix memory leak 1	Core Production
1h 6m 23s	gbooker (dev)	fix memory leak 2	Core Production

This data set was then analyzed to determine if the findings from the participant observation about the length of and participation in work episodes and deferral of work held in other settings.

Short and Individual Episodes

We found clear evidence of work being undertaken in short and individual episodes. First, Figure 1 shows that the mean and median duration of a task was shorter than 1 week. Second, as shown in Figure 2, approximately 80 percent of the tasks involved only a single participant writing code. Another 10 percent of tasks were primarily programmed by a single participant, with a small amount of polishing, such as fixing a spelling mistake, done by another participant. Less than 10 percent of tasks involved more than one person programming; these we call co-work. Even within the co-work episodes, only one involved any actions coded as management work to synchronize the work of two programmers. The co-work tasks showed no signs of systematically greater complexity, such as involving more lines of code.

Difficult Work Was Deferred

There was evidence that work was deferred when it seemed hard to complete. Figure 3 shows a plot of long running tasks. The release periods for each project are shown. The early actions in these tasks were all coded as support, usually

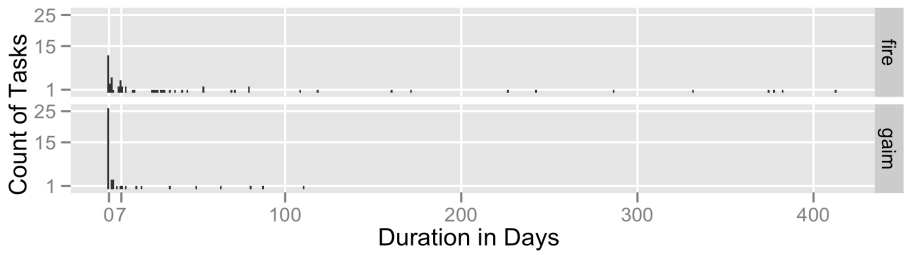

Figure 1. A Histogram of Task Length

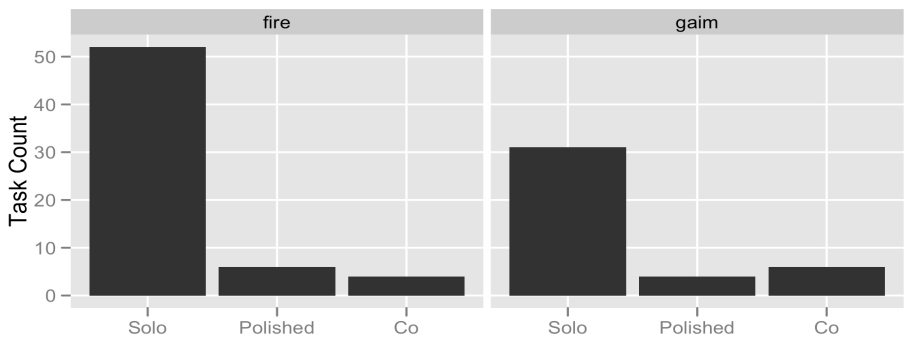

Figure 2. Tasks in Fire and Gaim Classified by the Number of Programmers in a Task (N = 106)

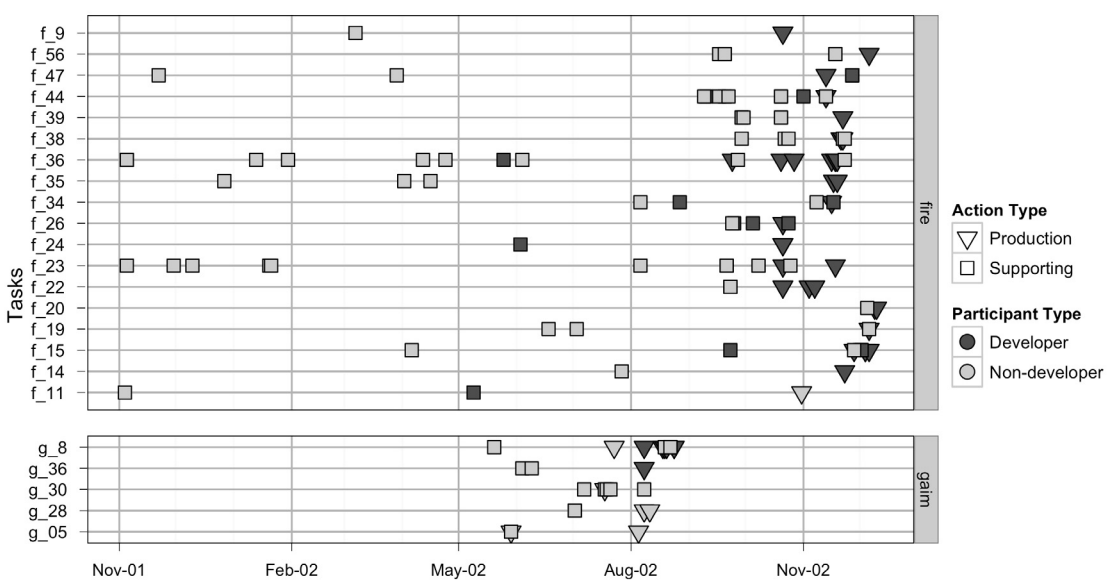

Figure 3. Action Time Lines for Long-Running Tasks (N = 22 of 106 total tasks)

feature requests or posts by non-developers demonstrating that a feature was desirable (light squares in Figure 3). Close inspection shows that all of the production work (dark triangles in Figure 3) for these tasks was completed relatively quickly at the end of the task, during the release period, even on those tasks that had been outstanding for months. This pattern is stronger for Fire than Gaim, which had fewer long running tasks, but present in both projects.

Qualitative investigation of these tasks provides evidence of deferral processes similar to those found in BibDesk. For example, *Task* f_9 in Figure 3 starts with a feature request made in March 2003. At that time, there is discussion among the developers of the desirability of the feature, 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 (e.g., a library to interface with Yahoo chat), facilitated waves of tasks, resolving outstanding acknowledged feature requests or bugs. These tasks are also striking for what did not occur: despite their early endorsement as desirable, there was no evidence of detailed planning, assignment, or breakdown of work toward these tasks, nor even explicit anticipation of a new library version. Rather the exogenous arrival of a new library prompted individual short, integrative tasks that built on this library and the existing codebase.

Overall, archival analysis of two additional projects supported the findings from the participant observation: the development work in Fire and Gaim was undertaken overwhelmingly through individual, short episodes of work. There was very little evidence of planning shared work (only found in a single task) and no evidence of resource management. Researchers have found such an absence of planning in small groups (Weingart 1992). They argue that performance under time pressure drives out planning, especially when the work is mutually visible. The FLOSS environment, however, is not one of deadlines and time pressures and we develop an alternative explanation below. Complex work appeared to be deferred, rather than being broken down into smaller components to be undertaken collaboratively.

Theory Development

This section develops a theory of how community-based FLOSS projects meet the challenge of completing complex work with volunteers at a distance: collaboration through open superposition. Our theory development proceeds in three parts. First, we develop theory to explain the patterns of work observed above. Second, we theorize as to how such an organization of work can result in complex software. Third, we theorize as to the socio-technical contingencies under which such organizing is more likely to be successful. Together these three elements allow us to generalize to theory (Yin 1994) from the replicated case studies above, through a process of abstraction and integration of existing theories. We follow this section with a discussion in which we demonstrate the usefulness of the theory to understand challenges for adapting the open source way in other domains.

Step 1: Explaining the Observed Patterns of Work in FLOSS

Our empirical studies revealed two patterns that need to be explained: the dominance of individual tasks over co-work tasks and the tendency to defer complex work rather than undertake co-work. To build our explanation we draw on two bodies of theory, the first regarding motivation for participation in FLOSS projects, the second regarding coordination of work.

Motivations for Contribution to FLOSS Projects

The willingness of participants to participate is the life-blood of a community-based project, just as operating capital is to a company: to thrive, a project has to continually attract the effort of potential participants to a coordinated whole. Understanding the motivation of participants is thus a key theme in FLOSS research. Ke and Zhang's (2010) work combining self determination theory and affective emotion theory provides an empirically supported model of motivation in open source projects. They begin with Ryan and Deci's (2000) spectrum of motivation, which extends the two well-known categories of motivation, intrinsic motivation (driven by interests and enjoyment of the individual) and extrinsic motivation (driven by rewards). Ryan and Deci argue that extrinsic motivation should be further broken down according to the locus of regulation involved, from controlled (by others) to autonomous (self-directed) as one approaches intrinsic motivation. Ke and Zhang found that FLOSS participants with more autonomous regulation produced greater task effort and greater persistence and consistency. Further, Ke and Zhang reconcile conflicting findings on the task effort effects of different motivations by proposing a moderating effect of satisfaction of psychological needs for competence, autonomy, and relatedness. Satisfaction of these needs within a project generates positive affect and, thus

> enhance individuals' expectations that their effort will lead to positive outcomes and make individuals more favorably evaluate the outcomes, which in turn leads individuals to expend a larger amount of effort (Ke and Zhang 2010, p. 786, drawing on Klein et al. 1999; Locke and Latham 2004).

Projects that are able to satisfy these needs are more likely to receive consistent task effort, intensifying the effort-producing effects of locus of control.

Viewed in the light of motivation theory, the experience of participant observation suggests that having adequate motivation depends on whether an actor expects the task to provide anticipated payoffs (intrinsic or extrinsic), the locus of regulation (from self to other), and whether the prospect of the task generates positive affect, which in turn depends on whether the actor's experience of the project provides an expectation of autonomy, competence, and relatedness. In this way, the organization of the work is directly linked to motivation and effort.

Consider, then, a developer deciding whether to work on a particular FLOSS development task. One possibility is that tasks are such that the motivation is entirely experiential, such as learning or enjoyment alone, and does not require that the task actually result in useful software (although useful software might enhance the experience). In that case, a developer may be sufficiently motivated to undertake the task even if other considerations suggest a lower likelihood of the resulting software being useful. Such motivations are consistent with the frequent finding that participants are motivated by the opportunity to learn (e.g., Lakhani and von Hippel 2003).

For other tasks, however, motivation will depend more strongly on the likelihood of developing useful code. Motivation may be bound up in the expected utility of the new code (e.g., a developer needing a new feature). Alternately, the developer may feel a psychological need for competence that can best be satisfied by the immediate creation of improved software. If such a task is one that a single developer can be reasonably sure to accomplish in the foreseeable future, the developer may again be motivated to undertake the task. The conduct of such individual tasks in a volunteer environment ensures a local locus of control while satisfying needs for autonomy.

Coordination of Software Development Tasks

Activating individual motivation, however, is only part of the story of successful FLOSS projects. While highly motivated, autonomous individuals can produce useful software by themselves (Krishnamurthy 2002), building sustained, successful, collective projects requires drawing together the work of many. Indeed, providing relatedness already requires working with others. Yet working productively together means working in a coordinated fashion. Coordination is defined as "managing dependencies between activities," where such dependencies generate the need to coordinate (Malone and Crowston 1994, p. 90). Coordination theory provides a modeling framework of actors performing tasks, where tasks might require or create resources of different kinds. A particular concern in software development is that a development task (task B) often requires the outputs of some other task (task A) before it can be performed, thus creating a task–task dependency. In other words, the likelihood of completing a task depends on the completion of necessary prerequisite tasks.

A common case is where the necessary code already exists as part of the project, that is, the prerequisite task A has already been performed. This situation describes simple sequential layering of independently motivated tasks: collaboration through superposition, as observed and described above. The layering of such tasks on the work of others, as well as the possibility of opportunistic support from others and the understanding of others in the project as audience, also provides relatedness in a manner that does not undermine autonomy or local locus of control. In the BibDesk episodes described above and depicted in Figure 4, this case describes the situation in the implementation of the "Container Column" (Panel A) and the implementation of "Web Groups," (Panel D). Each of these features built on what had come before, but each had its own, independently motivating payoff. More importantly, the work on which they relied, columns and groups, had earlier had their own independent motivational payoff. Neither columns nor groups were implemented just so that the "Container Column" or "Web Group" could be implemented.

The contrary situation is when the necessary code does not yet exist, that is, both tasks A and B still need to be performed. Such situations we call missing steps, shown in Panel B in Figure 4. The motivation for work on a task may be in place, but the groundwork is not in place and therefore its dependencies are not satisfied. In this situation, the challenge is to both attract individual effort and coordinate collective work, goals that are intimately linked and potentially in conflict. In the language of the studies above, actors who chose to work on these tasks will be engaged in co-work and would be reciprocally interdependent, both in terms of function and motivation. The difficulty is that co-work, depicted in Panel C in

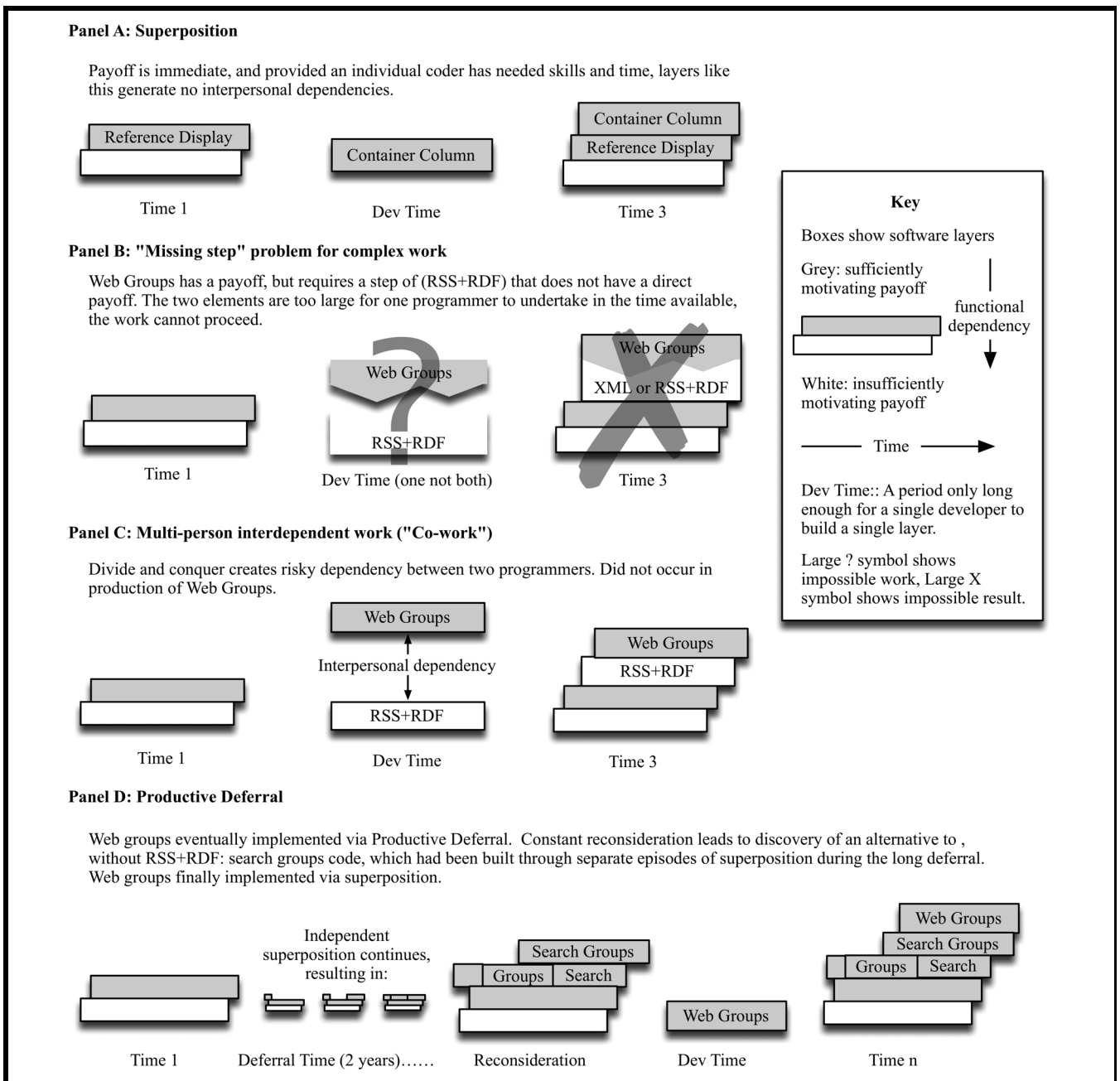

Figure 4. Different Dependency Structures for Work

Figure 4, creates the potential for a dependency problem: if one actor finishes task B but the other actor is unable to finish task A, then the first actor has wasted effort, as the success of task B depends on the completion of task A. The first actor is likely to experience negative affect derived from a loss of autonomy and competence, an experience that would undermine their future expectations and so undermine their continued motivation (Kiggundu 1983). The converse is also true: working on task A, which enables some functionality but does not deliver that functionality itself, may provide autonomy and competence, but no extrinsic reward absent completion of task B, which actually delivers the desired functionality. Given the uncertain reward, both developers will be reluctant to devote time to the tasks. Much the same

analysis applies to a single developer considering working on both tasks sequentially: any uncertainty about being able to finish both reduces the motivation to attempt either one. Both situations make other non-interdependent tasks relatively more attractive.

The deferred complex work encountered in both studies above has this character. In BibDesk, tasks such as removing dependence on the BibTeX format were waiting for the implementation of the BibDesk 2.0 framework. Without the framework in place, the format-independent system could not be implemented. The task had a motivation, but did not have its code dependencies satisfied by the existing codebase. Yet BibDesk 2.0 proceeded so slowly that expectations of its likely completion fell, reducing motivation for participating in building the framework. Nor was there sufficient expectation that the developers could collectively build a work breakdown with motivationally and functionally interdependent tasks to complete the work. As a result, the project members deferred the implementation of the feature until work undertaken for independent reasons rendered the original tasks able to be implemented through the superposition of layered individual work, routing around the missing step, as depicted in Panel D of Figure 4.

In summary, we have a fairly straightforward explanation of the observed working pattern: superimposed individual work is the predominant organization of tasks in FLOSS development because this type of work has the fewest dependencies and the simplest motivational situation. In particular, the superposition of individual work is more likely to be well motivated because it increases autonomy and competence without eliminating relatedness. Work that cannot be completed in this manner might be undertaken through co-work, if the tasks seem likely to be completed and the loss of autonomy is balanced by the increase in relatedness. If those conditions seem unlikely, as is often the case, the work is deferred until other work renders it achievable through the superimposition of individual work.

Step 2: How Collaboration Through Superposition Can Result in Complex Software

Given the technical and motivational environment of community open source projects, our reasoning above explains why most tasks have only a single individual programming and much complex work is deferred. What is more surprising, however, is that work done in this fashion can be so successful at producing useful software, or indeed, successful at all. Software is, after all, a highly complex artifact, giving rise to high levels of technical interdependence (e.g., Herbsleb et al. 2001). Why doesn't deferral of complex work simply lead to its abandonment?

If this argument is viewed in a wider context and asks how this limited type of work can lead to complex and functional software, the challenges come into focus: How does the project find a work breakdown that simultaneously satisfies the constraints of sequence, codebase, and motivation described above? Not only must a breakdown into a sequence of tasks be found (hard enough even for paid projects), but the tasks must all be adequately motivated, at just the right time, for people with available time and appropriate skills.

Imagine the challenge for a hypothetical manager of such a project. Not only would they have to (1) identify the outcomes and design a task sequence to implement them, but then (2) search out those not only able to perform them but willing to do so at (3) just the right time. Any effort toward these three meta-tasks is a cost to the project, thus these costs are search costs, a particular type of transaction cost (Williamson 1981). Thus, the question becomes: How can these projects accomplish such a search without crippling search costs?

We theorize that successful FLOSS projects achieve such a search at very low cost through the open availability of the software they produce. A fundamental feature of the FLOSS environment is that the software product itself is widely available, usually at zero financial cost. Indeed the application itself is really the first experience of a project for potential participants, who use the application in different daily activities (as described above in the participant observation). This use generates a set of perceived possible improvements, be they fixes to annoyances (such as the too-small screen described in the participant observation) or insight into ways the software could be extended, in a manner analogous to the resource-feedback described in discussion communities by Butler (2001). The tasks thus imagined are based on the current state of the artifact, a type of situated action (Suchman 1987). Brand (1995, p. vi) observes this as a feature of how buildings change over time, quoting an unnamed architect,

> Porches fill in by stages, not all at once, you know ….it happens that way because [the family] can always *visualize* the next stage based on what's already there.

This focus on what exists means the tasks that are identified are more likely to be sequentially appropriate, that is, achievable through the kinds of small, short, individual work layered on top of the current codebase that we observe in the studies above.

Of course the identification of tasks is only part of the search challenge facing projects: they must also identify developers who share that desire and who are capable. This identification is simple when a developer personally conceives an idea, but if a non-capable user has done so, a developer must be convinced and thus develop the motivation to undertake the task. As others have observed (e.g., Benkler 2002, p. 446), the information matching environment of FLOSS projects is rich and low cost, as such the feature-trackers and mailing lists of a project assist in the conduct of this search, coordinating capable actors with interesting ideas.

Successful community projects, we theorize, are based on the ability to discover a sequencing of motivated, layered tasks through a broad, situated search process facilitated by the usefulness of the product they produce and free-to-the-project distribution and communication infrastructures. Nonetheless, this search process is difficult and it is not hard to imagine finding projects in functional or motivational dead-ends, where desirable features are forever a missing layer away. No matter how widely an application is used, there might simply be no one with the motivation and skills to provide that layer. Arguably, shifting BibDesk away from a reliance on the BibTeX file format is an example of this problem. Rather than seek an interdependent work breakdown, as might be found in a commercial software project, we theorize that the open, situated search process is complemented by productive deferral.

Productive Deferral

We have argued that work that is conceptualized but fails to have its dependencies met or a motivated actor at the time of conceptualization would be deferred. Deferring a desired but difficult feature does not take available time away from other development; the layering of other small, independently motivated, layers can proceed. These tasks alter the codebase over time, allowing participants to periodically reconceptualize the work needed, acting when its dependencies are satisfied. This situation was encountered by the first author in participant observation and illustrated by the earlier example in which the founder of BibDesk initially perceived the desirable task as too much work but considered it from time to time until code—written for entirely different reasons—made the task easy enough to undertake through relatively simple, quick, and individual work.

Yet couldn't the deferral of work just as easily make its eventual accomplishment harder, rather than easier? This outcome seems especially likely if changes over the deferral time make the application and codebase substantially different from what they were when the feature was conceptualized. We theorize that there is a bias toward complex tasks getting easier rather than harder because the intermediate layers—the work that is possible—tend to be smaller units of functionality. These small units are more easily understood and reused by others. This approach reverses the typical software engineering practice of re-factoring (e.g., Fowler and Beck 1999) that seeks an architecture of small, reused pieces. Rather than achieving this through *post hoc* analysis and rewriting of feature-rich code, such an architecture is achieved in process through constraints that restrict the creation of large chunks of code. Architectures of small layers generate future value because they provide a reconfigurable set of services that can be composed into higher-level functions, generating longer-term option value (Baldwin and Clark 2001). Smaller layers generate option value because they provide the option, but not the requirement, that they be reconfigured into higher-level functions.

Moreover, an evolving architecture of small layers can be more easily observed and understood by others over time, facilitating review and boosting the process of situated task conceptualization. Dabbish et al. (2011) describe this in the radical transparency afforded by GitHub, using interview data to show that the visibility of the work of others in smaller pieces provides ideas, motivation, and knowledge to collaborations. Similarly Colfer and Baldwin (2010) identify the importance of actionable transparency in the organization of open source production, and Boudreiu et al. (2011) highlight the importance of understandable contributions in TopCoder. If the work that occurred during a deferral was made up of large chunks of code, hard to reuse and hard to understand, deferral would not be productive in the way we describe.

Thus we theorize that the forced constraints of community open source projects promote working in small layers, observed by others over time. This way of working generates an architecture that is more reusable, of higher quality, and more easily understood and which, therefore, tends to make deferred work easier over time, rather than harder.

Figure 5 attempts to illustrate the theory of collaboration through open superposition. At the center is the codebase, with the user community depicted on the left-hand side. Activity in a growing and active user community "throws off" both situated improvement ideas and, more rarely, attracts new developers. The right-hand side depicts the project developers considering the stock of suggested improvement ideas in the context of the updated codebase, deferring difficult ideas for future consideration or producing a new layer to

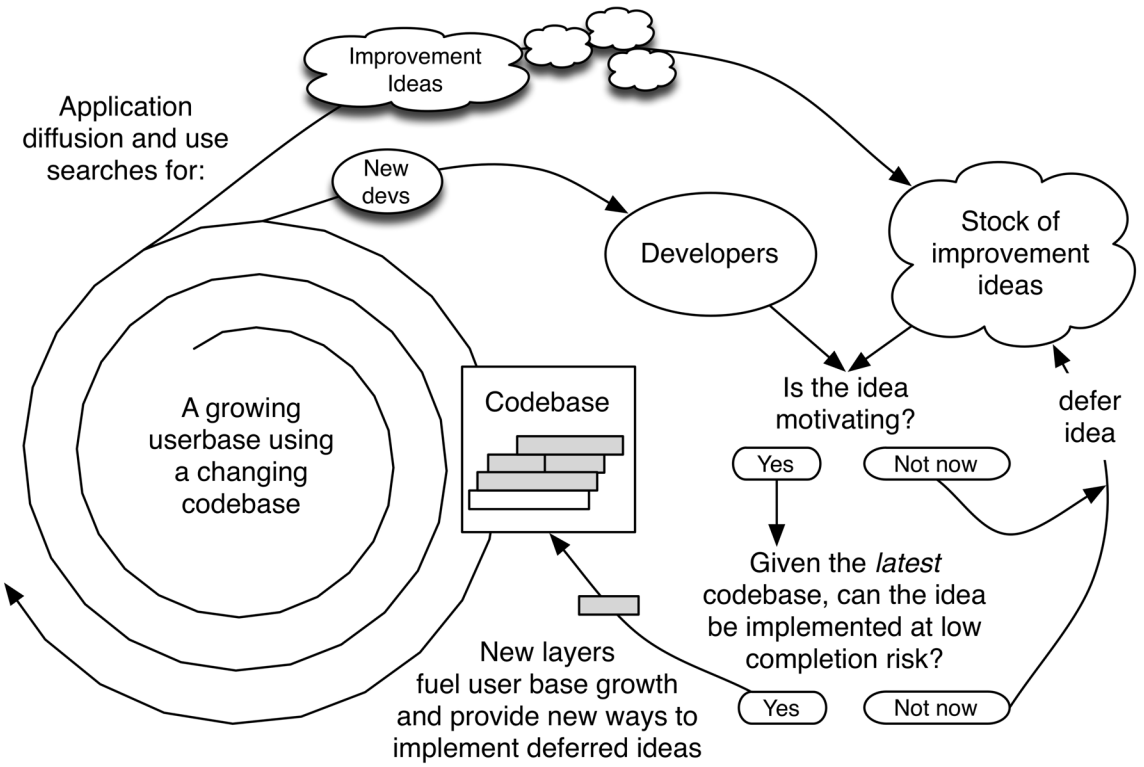

Figure 5. Collaboration Through Open Superposition

be superimposed onto the existing codebase. The new layer contributes to constructive feedback loops on both sides. For users, increased functionality drives more use (leading to more sequentially appropriate ideas and, sometimes, more developers). For developers, new layers provide an opportunity to discover a new, simpler way to implement a deferred idea (thus adding yet another layer and continuing the positive feedback).

Step 3: Identification of Conditions under Which Collaboration Through Open Superposition Is Likely to Be Successful

We have argued that core to the open source way in community-based FLOSS is the superposition of individual tasks together with an open application-driven search that coordinates a work breakdown and task assignment that aligns with the motivations of potential participants. Such an approach is surprisingly effective. Moreover, it appears to succeed precisely where other organizational or management approaches face difficulties: undertaking functionally interdependent work, with sporadically available volunteers, at a distance.

For those seeking to learn from and adapt this way of working, it is critical to understand the conditions under which it is likely to be effective. This section draws on our theory to examine these conditions and uses them to discuss the potential and the difficulties likely to be faced when the open source way is adapted to other domains. The conditions fall into three categories: (1) attributes of the object of work, (2) irrevocable openness, and (3) time.

Attributes of the Object of Work

The object of work—that which is worked on—is key to collaboration through open superposition. In the case of FLOSS, that object is software. We identify three attributes that are associated with collaboration through open superposition and which are realized in software. These attributes are (1) layerability, (2) low instantiation costs, and (3) low distribution costs. While these attributes are not unique to software, or even to information artifacts per se, the extent that they are available in other work will directly affect the success of adapting collaboration through open superposition.

Layerability: At the core of collaboration through superposition is the ability to accomplish complex collective work in a sequence of patches or layers, each of which has its own payoffs and does not depend on future work for its usefulness. Some objects of work have this characteristic, while many do not. Buildings, for example, are frequently improved in layers, such as the ziggurats of Mayan history (where the layering occurred over hundreds of years) or the year by year improvements made to single family homes by their owners (Brand 1995).

Layers, in this sense, are different from generic steps because each layer creates an (adequately) finished artifact. For example, an airplane certainly can be built in steps: first the fuselage, then the engines, and finally the wings. But until it is complete, none of these steps provide any utility payoff, at least not in the sense of a flying plane. Small layers added to an existing base, each with its own sufficient instrumental payoff (forming "stackable incentives"[3]), are core to understanding when collaboration through open superposition is appropriate.

Low Instantiation Costs: Instantiation costs are the costs of moving from a design to a useful artifact (and not the process of creating the design). When instantiation costs are high (such as moving from a blueprint to a finished house), the use value of the changes is very hard to realize, undermining a key type of motivation seen in FLOSS projects. Low instantiation costs are very important for collaboration through open superposition: if it is expensive to rebuild the existing work to place a new layer upon it, then adding that layer is itself very expensive.

Low Distribution Costs: The Internet has drastically reduced distribution costs for software; indeed, distribution is often free to the project thanks to hosting sites such as SourceForge, GitHub or Google Code. Prior to Internet software delivery, updates involved printing CDs (or copying tapes) and shipping them to customers, a much more expensive proposition that occurred much less frequently. Low distribution costs are key in three ways: (1) they are complementary to low instantiation costs in that distribution must occur before use, (2) they reduce barriers for new users to adopt the software, growing the user community and, most importantly, (3) rapid and inexpensive distribution allows new layers to spread out quickly to users. Rather than wait for periodic shipping periods, these layers are quickly in the hands of developers generating interesting, new, and situated ideas for contributions.

[3]We are indebted to a colleague at a conference for this pithy phrase. Unfortunately, despite many attempts, we have been unable to identify them and thank them by name.

These three attributes of the object of work are conceptually separate but all necessary for collaboration through open superposition to operate at the scale it does today. While other collective activities, such as buildings, have resulted from the superposition of functional and independently motivated layers, these physical products have very high instantiation and distribution costs. Software, on the other hand, has all three characteristics: it is as though any building painstakingly constructed anywhere were instantly relocatable without cost and available to all to base their own building upon.

Irrevocable Openness

The attributes of the object of work described above are necessary for collaboration through open superposition, but they are not sufficient. Software, or other artifacts, may be layerable as well as widely and cheaply available, but legally encumbered such that they cannot be used, redistributed, or built upon. Irrevocable openness describes a technical, legal, and ethical situation that is also necessary for the success of superposition. While the technical difficulty of "recalling" a contribution once it is widely available is substantial, legal guarantees are more important. Stallman's "four freedoms" encode this openness explicitly in software licenses: the freedom to (1) run for any purpose, (2) change the software, (3) redistribute the application, and (4) redistribute any changes you make. These kinds of openness are all important. Without the ability for many to run the application and distribute changes, the application-led search process cannot operate and search costs for interested and motivated developers would be strongly increased. Without the ability to change existing code, alignments of interest and motivation may pass without becoming a contribution or the basis for others' work.

The more irrevocable the openness, the better the conditions are for collaboration through open superposition. If a contributor is free to remove their layers, then all subsequent work is not superposition but a special kind of co-work (because the layers are not motivationally independent). Each layer depends on continued non-revocation of its foundation, a long-term personal interdependency. Open source licenses guarantee non-revocability of contributions, either explicitly (e.g., Apache) or implicitly (through an absence of conditions on the copyright grant). This avoids what Heller and Eisenburg (1998) called "the tragedy of the anticommons," discussing chilling of medical research due to patents, where researchers are unwilling to work in areas covered by patents since their work could become inaccessible if the patent holder ever wished to enforce their ownership rights.

Irrevocable openness means that even if a developer were to regret the decision to contribute, their contribution would remain freely available and be able to support layers built on it. Therefore, developers do not have to hope for continued cooperation from others. To the extent that those seeking to adapt the open source way are not able to create an expectation of the irrevocability of work that has gone before, or incur substantial contract monitoring transaction costs in an effort to do so, collaboration through open superposition is less likely to be effective.

Time

Collaboration through open superposition takes time for a number of different reasons. The open search process through which a simultaneous solution to the work breakdown and task assignment problems is found takes time. It takes time to wait for an application to diffuse through a potential user community, generating the continual advertisement that brings contributors to a project. It takes time when a project faces a missing step problem and important features are deferred, to wait for other work to provide the layers that render the deferred work easier. To the extent that time is not available for these processes to play out, collaboration through open superimposition will be less appropriate as a way to manage work.

One counterintuitive source of time pressure is the availability of financial investment. Investment—even nonprofit-oriented investment—has opportunity costs, generating the time-cost of money and pressure to see payoffs sooner rather than later. Thus a subtle impact of the very low financial requirements of open source infrastructure—often free to the project—is that it removes a requirement for investment and investment and the deadlines that result. When time is not available and deadlines loom, there is urgency to increase the pace of development. As we illustrate below with Apple's development of the Safari browser, the techniques available to speed up development largely cut against the open source way we are describing.

Together, these three socio-technical features work to undergird collaboration through open superposition and thus the open source way. Without layerability, low instantiation, and low distribution costs, the creation and distribution of independently motivated patches is not possible. Without irrevocable contributions, participants cannot build on each other's contributions with confidence. Without openness, an application cannot be its own advertisement and the search process, matching task conceptualization with availability and motivation, is threatened. Without the ability to wait, at little or no cost, these motivated tasks are less likely to fall in the right order or "route around" the blockages of complex work.

Discussion

Challenges for Adaptation

Much of the interest in FLOSS and its development stems from the difficulties encountered in IS development, even when colocated, together with difficulties encountered in distributed work generally, especially when crossing organizational boundaries. FLOSS seems to solve these two difficulties by combining them and does so without needing financial investment—a truly remarkable achievement, raising the hope that many useful lessons for conventional development can be extracted from the FLOSS model of organizing (Agerfalk and Fitzgerald 2008; von Krogh and von Hippel 2006). Yet the theory presented in this paper suggests limits to this wider applicability.

Aims for adaptability in for-project IS function have taken two main forms. The first is sometimes known as "inner source" (Dinkelacker et al. 2002), where a firm attempts to generate an open source community within its corporate boundaries (examples include HP and the U.S. Department of Defence's forge.mil). The second is to integrate FLOSS components into the firm's IS strategy, both for internal IS and for the production of IS for sale (Agerfalk and Fitzgerald 2008).

The inner source strategy presents significant adaptation challenges. It is relatively simple to replicate the IT infrastructure of FLOSS inside a corporation; indeed, selling such systems was part of the business models of Sourceforge and Collab.net. Yet simple importation of technology has not been sufficient to replicate a socio-technical phenomenon, an outcome that is unsurprising in light of the history of IS scholarship (e.g, DeSanctis and Poole 1994; Orlikowski 1992). Existing research has pointed out a set of reasons why adaptation in this way is difficult, focusing on process, difficulties of openness in corporate culture, architecture, and de-emphasizing incentives such as learning and fun (Gaughan et al. 2009; Gurbani et al. 2006).

Our work highlights an additional factor: the usefulness of productive deferral is undermined because firms inherently face deadlines due to up-front investment. To what extent can a firm delay implementation of a desired, potentially profitable feature in the hope that it will become easier? Given this, inner source seems most likely to work in two circumstances. The first is those firms willing and able to

sustain a "free time" culture of exploration and learning, such as "20 percent time" at Google (and earlier at HP). The second are those firms that have significant non-marketed infrastructure needs, which if met can save sufficient money to generate enough resources to pay the developers. In this second case, however, limiting the community to just those inside the corporation does not seem necessary. Extending such communities beyond the boundaries of the firm has seen success in projects such as IBM's founding and extension of the Eclipse community (Wagstrom 2009).

A second strategy is for firms to adapt FLOSS to their IS function. This approach is particularly challenging if the firm's strategy requires them to actively develop the code, rather than passively consume it (Fitzgerald 2006; Shah 2006). Our theory suggests that success will depend on the extent to which the firm can afford to align with work undertaken through small layers and deferral of complex work or whether their market imperatives generate deadlines and a strategic need for secrecy—an element of risk not faced by individuals considering involvement.

The case of Apple's Safari browser illustrates these difficulties. Safari is based on the open source khtml project. Apple's market strategy for Safari called for high secrecy during product development, and market pressures placed a strong premium on rapid time to market. By taking the work in-house, Apple was able to move secretly and much more quickly than the khtml project, short-circuiting slow processes of layering and deferral. When Apple released Safari, they released their source code modifications and announced a desire to work with the khtml community in future, sharing on-going maintenance and development costs. Yet the members of the khtml project were displeased, as illustrated by the following quotation:

> Do you have any idea how hard it is to be merging between two totally different trees when one of them doesn't have any history? That's the situation KDE is in. We created the khtml-cvs list for Apple, they got CVS accounts for KDE CVS. What did we get? We get periodical code bombs in the form of them releasing WebCore.…They made a conscious decision about not working with KDE developers. All I'm asking for is that all the clueless people stop talking about the cooperation between Safari/Konqueror developers and how great it is. There's absolutely nothing great about it. In fact "it" doesn't exist (Source: https://blogs.kde.org/2005/04/28/so-when-will-khtml-merge-all-webcore-changes).

Apple's modifications were too large and had branched from khtml too long ago for them to be easily integrated; Apple's work had not proceeded in observable, short, and small layers. This story stands in strong contrast to IBM's adoption of Apache's httpd web server. For IBM there was no need for secrecy, since IBM was generating its revenue from services and higher value added software. Further, the httpd server was already capable enough that IBM did not feel the need for radical innovation and could instead develop in the FLOSS model of small, visible steps. Corporate involvement of this type might even attract additional volunteers, as recently found in a study of corporate impact on the Gnome community (Wagstrom et al. 2010), possibly by providing layers that make the work of volunteers easier. On the other hand, firms seeking to push the pace of projects forward by taking complex work in-house, especially in secret, and satisfying open source licenses through periodic large code releases should expect to face significant difficulties.

Conclusion and Contribution

The theory and empirical work presented in this paper makes useful and significant contributions, albeit not without limitations. The primary limitation is the decision to trade empirical generalizability beyond three specific FLOSS cases from the mid-2000s for the depth needed for theory development. For two cases, we only studied relatively short periods, balancing this against the four years of observation in BibDesk. Moreover, we chose to study only community-based projects with no paid participants, rather than very large, high-profile projects such as Apache httpd or the Linux kernel, or other projects that include both volunteer participants and those paid by companies for their involvement. On the one hand, corporate involvement would seem to open up new possibilities, as companies can motivate developers with pay and provide coordination for groups of developers working on a new feature. On the other hand, the results of paid work and the concerns of companies (e.g., the need for particular features at a particular time or concerns about wasted developer time) may be incompatible with the processes and motivations of open superposition as we described them. An important topic for future research is to reveal how such tensions are handled in successful hybrid projects.

Even though there are limitations, this 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 by providing a socio-technical theory of organizing where the

detailed attributes of information technology artifacts play a central role. 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. Moreover, the work suggests that material characteristics of the artifacts made from information are important in the success of FLOSS projects. This implies that efforts to learn from FLOSS teams for virtual teams and distributed work in general ought to focus on changing task work as well as team work. From our perspective, the lessons of FLOSS are more about work redesign than about how to run an effective virtual team, about task work in context rather than generic team work processes. Rather than look to FLOSS for lessons on, say, conflict management at a distance, efforts could focus on redesigning the work of virtual teams to benefit from layerability, open distribution, irrevocable contribution, and time. This requires a detailed socio-technical study of the specific task work and its motivational and material contexts, as epitomized by Trist and Bamforth (1951).

A contribution is also made to Organizational Science because a new model of organizing is described and analyzed. While future work ought to explore this more fully, collaboration through open superposition is distinct from organizing via hierarchies, markets, or networks (Benkler 2002; Powell 1990). It is distinct from hierarchies in that work is not directed and planned from above; rather, work choices are made with high levels of autonomy and little planning. It is also distinct from organizing common to firms in that there is no system that defers and pools payoffs, as do capital investment and employee salaries; rather, the work is mostly constrained to that which has an immediate motivating payoff. Collaboration through open superposition is also distinct from a market because while it conducts a type of search it lacks a pricing mechanism; rather than discovering a price at which potential developers will work, open superposition creates opportunities for work that provide a set of nonmonetary payoffs. These, like barter systems, can be "chunky," failing to match opportunities and developers, leading to deferral. Open superposition is also distinct from networks, which thrive on what Powell (1990) called "relational contracting," highly embedded in cultural and geographic institutions, since FLOSS participants rarely know each other in advance and produce while minimizing interdependency. Finally, our model is distinct from, but complementary to, Benkler's (2002) commons-based peer production, because we focus on the sequencing of project work and identify the role of productive deferral and artifact-led situated search. Open superposition is, perhaps, closer to the organization of knowledge production in "the republic of science" (Polanyi 1962) in that motivationally independent contributions build on a growing base over time, with the difference being that open superposition creates artifacts which themselves then play a role in the conceptualization of tasks and the search for those motivated to undertake them.

Finally, a contribution is made to the literature on motivation and job design by showing a type of work that facilitates autonomy and local locus of control while providing a type of relatedness that does not undermine either. Moreover, this paper shows how a coordinated work breakdown for tasks of this type might be achieved: through its discovery in work over time through productive deferral, rather than an active, planned design.

This paper began with the observation that the success of FLOSS and other forms of open collaboration is surprising because they face three well-known challenges to organizing: complex work, working at a distance, and working with volunteers. Working at a distance, outside formal organizations, already sacrifices many traditional sources of control and motivation. Relying on self-motivated volunteers reduces the relevance of these, but creates the challenge of trying to find a way to organize that draws together relatively independent work into a cohesive, complex, and valuable whole, while maintaining a fertile ground for volunteerism. The argument of this paper is that a combination of characteristics found in having artifacts made from information as the objects being built, together with irrevocable openness and time, can be a solution to these challenges, providing the bedrock on which the superposition of small, independently motivated layers over time in an open way can build valuable interdependent artifacts and provide mutual inspiration, albeit at the not-inconsiderable cost of uncertain delay.

Collaboration through open superposition might be frustratingly slow and uncertain from a traditional management perspective that seeks to do more with known, expensive, but coercible, resources. Yet if the main challenge is to "outsource to an unknown workforce" (Agerfalk and Fitzgerald 2008), this way of working makes clear sense. Understanding the core of the open source way, and how it is linked to characteristics present in artifacts made from information as objects of work, is vital to pursuing successful adaptation or hybridization and learning from the surprisingly successful new ways of collaborating associated with the rise of widespread information systems.

Acknowledgments

The authors would like to thank the members of the Syracuse FLOSS research team, including Hala Annabi, Robert heckman, Chengetai Masango, Kangning Wei, Quing Li, Yeliz Eseryl, and

Andrea Wiggins. We would also like to thank members of the FLOSS community who gave significant time discussing and debating the open source way. Thanks also to Jim Herbsleb, Patrick Wagstrom, and the editors and anonymous reviewers.

This research was partially supported by the U.S. National Science Foundation, under grant numbers 03–41475, 04–14468, 05–27457, and 07–08437.

References

Agerfalk, P. J., and Fitzgerald, B. 2008. "Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy," *MIS Quarterly* (32:2), pp. 385-409.

Bailey, D. E., and Barley, S. R. 2005. "Return to Work: Towards Post-Industrial Engineering," *IIE Transactions* (37:8), pp. 737-752.

Baldwin, C. Y., and Clark, K. B. 2000. *Design Rules: The Power of Modularity*, Cambridge, MA: Harvard Business School Press.

Baldwin, C. Y., and Clark, K. B. 2006. "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?," *Management Science* (52:7), pp. 1116-1127.

Barley, S. R. 1986. "Technology as an Occasion for Structuring: Observations on CT Scanners and the Social Order of Radiology Departments," *Administrative Science Quarterly* (31:1), pp. 78-108.

Barley, S. R., and Kunda, G. 2001. "Bringing Work Back In," *Organization Science* (12:1), pp. 76-95.

Benkler, Y. 2002. "Coase's Penguin, or, Linux and The Nature of the Firm," *Yale Law Journal* (112), pp. 369-446.

Boudreau, K. J., Lacetera, N., and Lakhani, K. R. 2011. "Incentives and Problem Uncertainty in Innovation Contests: An Empirical Analysis," *Management Science* (57:5), pp. 843-863.

Bowker, G. C., and Star, S. L. 2000. *Sorting Things Out: Classification and Its Consequences*, Cambridge, MA: MIT Press.

Brand, S. 1995. *How Buildings Learn: What Happens After They're Built*, New York: Penguin Books.

Butler, B. S. 2001. "Membership Size, Communication Activity, and Sustainability: The Internal Dynamics of Networked Social Structures," *Information Systems Research* (12:4), pp. 346-362.

Capiluppi, A., and Michlmayr, M. 2007. "From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects," in *Open Source Development, Adoption and Innovation*, The International Federation for Information Processing (IFIP), J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti (eds.), New York: Springer, pp. 31-44.

Colfer, L., and Baldwin, C. Y. 2010. "The Mirroring Hypothesis: Theory, Evidence and Exceptions," Working Paper No. 10-058, Harvard Business School Finance.

Conley, C. A., and Sproull, L. 2009. "Easier Said than Done: An Empirical Investigation of Software Design and Quality in Open Source Software Development," in *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences*, Los Alamitos, CA: IEEE Computer Society Press.

Crowston, K., and Wade, M. 2010. "Introduction to JAIS Special Issue on Empirical Research on Free/Libre Open Source Software," *Journal of the Association for Information Systems* (11:11), pp. i-v.

Crowston, K., Wei, K., Howison, J., and Wiggins, A. 2012. "Free/Libre Open Source Software Development: What We Know and What We Do Not Know," *ACM Computing Surveys* (44:2), Article 7.

Dabbish, L., Stewart, C., Tsay, J., and Herbsleb, J. D. 2011. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, Seattle, WA, February 11-15, pp. 1277-1286.

Daniel, S. L., and Diamant, E. I. 2008. "Network Effects in OSS Development: The Impact of Users and Developers on Project Performance," in *Proceedings of the 29th International Conference on Information Systems*, Paris, France, December 14-17, Paper 122.

Dennis, A. R., Valacich, J. S., and Fuller, R. M. 2008. "Media, Tasks, and Communication Processes: A Theory of Media Synchronicity," *MIS Quarterly* (32:3), pp. 575-600.

DeSanctis, G., and Poole, M. S. 1994. "Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory," *Organization Science* (5:2), pp. 121-147.

Dinkelacker, J., Garg, P. K., Miller, R., and Nelson, D. 2002. "Progressive Open Source," in *Proceedings of the 24th International Conference on Software Engineering*, Orlando, FL, May 19-25, pp. 177-184.

Dunlop, J. J. 1990. "Balancing Power: How to Achieve a Better Balance Between Staff and Volunteer Influence," *Association Management* (January).

Duval, J. 2010. *Next Generation Democracy: What the Open-Source Revolution Means for Power, Politics, and Change*, New York: Bloomsbury USA.

Fitzgerald, B. 2006. "The Transformation of Open Source Software," *MIS Quarterly* (30:3), pp. 587-598.

Fowler, M., and Beck, K. 1999. *Refactoring: Improving the Design of Existing Code*, Boston: Addison-Wesley Professional.

Gallivan, M. J. 2001. "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), pp. 277-304.

Gaughan, G., Fitzgerald, B., and Shaikh, M. 2009. "An Examination of the Use of Open Source Software Processes as a Global Software Development Solution for Commercial Software Engineering," in *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*, Patras, Greece, August 27-29, pp. 20-27.

Geiger, R. S., and Ribes, D. 2011. "Trace Ethnography: Following Coordination through Documentary Practices," in *Proceedings of the 44th Hawaii International Conference on System Sciences*, Los Alamitos, CA: IEEE Computer Society Press.

Gurbani, V. K., Garvert, A., and Herbsleb, J. D. 2006. "A Case Study of a Corporate Open Source Development Model," in

Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, May 20-28, pp. 472-481.

Hahn, J., Moon, J. Y., and Zhang, C. 2008. "Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties," *Information Systems Research* (19:3), pp. 369-391.

Handy, C. 1988. *Understanding Voluntary Organizations*, New York: Penguin Books.

Harvey, L. J., and Myers, M. D. 1995. "Scholarship and Practice: The Contribution of Ethnographic Research Methods to Bridging the Gap," *Information Technology & People* (8:3), pp. 13-27.

Heller, M. A., and Eisenberg, R. S. 1998. "Can Patents Deter Innovation? The Anticommons in Biomedical Research," *Science* (280:5364), pp. 698-701.

Herbsleb, J. D., Mockus, A., Finholt, T. A., and Grinter, R. E. 2001. "An Empirical Study of Global Software Development: Distance and Speed," in *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, May 12-19, pp. 81-90.

Hertel, G. 2007. "Motivating Job Design as a Factor in Open Source Governance," *Journal of Management and Governance* (11:2), pp. 129-137.

Hinds, P. J., and Mortensen, M. 2005. "Understanding Conflict in Geographically Distributed Teams: The Moderating Effects of Shared Identity, Shared Context, and Spontaneous Communication," *Organization Science* (16:3), pp. 290-307.

Ives, B., Hamilton, S., and Davis, G. B. 1980. "A Framework for Research in Computer-Based Management Information Systems," *Management Science* (26:9), pp. 910-934.

Jarvenpaa, S. L., and Leidner, D. E. 1999. "Communication and Trust in Global Virtual Teams," *Organization Science* (10:6), pp. 791-815.

Ke, W., and Zhang, P. 2010. "The Effects of Extrinsic Motivations and Satisfaction in Open Source Software Development," *Journal of the Association for Information Systems* (11:12), pp. 784-808.

Kiggundu, M. N. 1983. "Task Interdependence and Job Design: Test of a Theory," *Organizational Behavior and Human Performance* (31:2), pp. 145-172.

Kittur, A., and Kraut, R. E. 2008. "Harnessing the Wisdom of Crowds in Wikipedia: Quality Through Coordination," in *Proceedings of the 2008 ACM Conference on Computer-Supported Cooperative Work*, San Diego, CA, November 8-12, pp. 37-46.

Klein, H. J., Wesson, M. J., Hollenbeck, J. R., and Alge, B. J. 1999. "Goal Commitment and the Goal-Setting Process: Conceptual Clarification and Empirical Synthesis," *Journal of Applied Psychology* (84:6), pp. 885-896.

Knorr-Cetina, K. 1999. *Epistemic Communities*, Cambridge, MA: Harvard Education Press.

Krishnamurthy, S. 2002. "Cave or Community: An Empirical Examination of 100 Mature Open Source Projects," *First Monday* (7:6).

Lakhani, K., and von Hippel, E. 2003. "How Open Source Software Works: 'Free'' User-to-User Assistance," *Research Policy* (32:6), pp. 923-943.

Leonardi, P. M. 2010. "Digital Materiality? How Artifacts Without Matter, Natter," *First Monday* (15:6-7).

Leonardi, P. M., and Barley, S. R. 2008. "Materiality and Change: Challenges to Building Better Theory about Technology and Organizing," *Information and Organization* (18:3), pp. 159-176.

Lewis, I. M. 1985. *Social Anthropology in Perspective: The Relevance of Social Anthropology*, Cambridge, UK: Cambridge University Press.

Lipnack, J., and Stamps, J. 1997. *Virtual Teams: Reaching Across Space, Time and Organizations with Technology*, Hoboken, NJ: John Wiley and Sons, Inc.

Locke, E. A., and Latham, G. P. 2004. "What Should We Do About Motivation Theory? Six Recommendations for the Twenty-First Century," *Academy of Management Review* (29:3), pp. 388-403.

MacCormack, A., Rusnak, J., and Baldwin, C. Y. 2006. "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Management Science* (52:7), pp. 1015-1030.

Malhotra, A., and Majchrzak, A. 2012. "How Virtual Teams Use Their Virtual Workspace to Coordinate Knowledge," *ACM Transactions on Management Information Systems* (3:1), pp. 1-14.

Malone, T. W., and Crowston, K. 1994. "The Interdisciplinary Theory of Coordination," *ACM Computing Surveys* (26:1), pp. 87-119.

Michlmayr, M. 2004. "Managing Volunteer Activity in Free Software Projects," in *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, Boston, June 27-July 2, pp. 39-50.

Myers, M. 1999. "Investigating Information Systems with Ethnographic Research," *Communications of the Association for Information Systems* (2), Article 23.

O'Mahony, S., and Ferraro, F. 2007. "Governance in Collective Production Communities," *Academy of Management Journal* (50:5), pp. 1079-1106.

Olson, G. M., and Olson, J. S. 2000. "Distance Matters," *Human–Computer Interaction* (15:2), pp. 139-179.

Orlikowski, W. J. 1992. "Learning from Notes: Organizational Issues in Groupware Implementation," in *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, Toronto, Canada, October 31-November 4, pp. 362-369.

Orlikowski, W. J., and Iacono, C. S. 2001. "Research Commentary: Desperately Seeking the 'IT' in IT Research: A Call to Theorizing the IT Artifact," *Information Systems Research* (12:2), pp. 121-134.

Orlikowski, W., and Scott, S. V. 2008. "Sociomateriality: Challenging the Separation of Technology, Work and Organization," *The Academy of Management Annals* (2:1), pp. 433-474.

Parnas, D. L., Clements, P. C., and Weiss, D. M. 1981. "The Modular Structure of Complex Systems," *IEEE Transactions on Software Engineering* (11:3), pp. 259-266.

Pentland, B. T. 1999. "Building Process Theory with Narrative: From Description to Explanation," *Academy of Management Review* (24:4), pp. 711-724.

Polanyi, M. 1962. "The Republic of Science: Its Political and Economic Theory," *Minerva* (1:1), pp. 54-74.

Powell, W. W. 1990. "Neither Market Nor Hierarchy: Network Forms of Organization," *Research in Organizational Behavior* (12), pp. 295-336.

Rousseau, V., Aube, C., and Savoie, A. 2006. "Teamwork Behaviors: A Review and an Integration of Frameworks," *Small Group Research* (37:5), pp. 540-570.

Ryan, R. M., and Deci, E. L. 2000. "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions," *Contemporary Educational Psychology* (25:1), pp. 54-67.

Sarker, S., and Sahay, S. 2004. "Implications of Space and Time for Distributed Work: An Interpretive Study of US–Norwegian Systems Development Teams," *European Journal of Information Systems* (13:1), pp. 3-20.

Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. 2006. "Guest Editorial: Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice* (11), pp. 95-105.

Shah, S. K. 2006. "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science* (52:7), pp. 1000-1014.

Shirky, C. 2008. *Here Comes Everybody: The Power of Organizing Without Organizations*, New York: Penguin Press.

Stewart, K. J., and Gosain, S. 2006. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *MIS Quarterly* (30:2), pp. 291-314.

Suchman, L. A. 1987. *Plans and Situated Actions: The Problem of Human–Machine Communication* (2nd ed.) Cambridge, UK: Cambridge University Press.

Trist, E. L., and Bamforth, K. W. 1951. "Social and Psychological Consequences of the Longwall Method of Coal-Getting," *Human Relations* (4:1), pp. 3-38.

Van Maanen, J. 1988. *Tales of the Field: On Writing Ethnography*, Chicago: University of Chicago Press.

Von Krogh, G., and von Hippel, E. 2003. "Editorial: Special Issue on Open Source Software Development," *Research Policy* (32:7), pp. 1149-1157.

Von Krogh, G., and von Hippel, E. 2006. "The Promise of Research on Open Source Software," *Management Science* (52:7), pp. 975-983.

Wagstrom, P. 2009. "Vertical Communication in Open Software Engineering Communities," unpublished Ph.D. dissertation, Carnegie Mellon University.

Wagstrom, P., Herbsleb, J. D., Kraut, R. E., and Mockus, A. 2010. "The Impact of Commercial Organizations on Volunteer Participation in an Online Community," presentation at the Academy of Management Conference (OCIS Division), Montréal, Canada, August 6-10.

Walsham, G., and Sahay, S. 1999. "GIS for District-Level Administration in India: Problems and Opportunities," *MIS Quarterly* (23:1), pp. 39-65.

Watson-Manheim, M. B., Chudoba, K. M., and Crowston, K. 2002. "Discontinuities and Continuities: A New Way to Understand Virtual Work," *Information, Technology and People* (15:3), pp. 191-209.

Weick, K. E. 1989. "Theory Construction as Disciplined Imagination," *Academy of Management Review* (14:4), pp. 516-531.

Weick, K. E. 1995. "What Theory Is Not, Theorizing Is," *Administrative Science Quarterly* (40:3), pp. 385-390.

Weingart, L. R. 1992. "Impact of Group Goals, Task Component Complexity, Effort, and Planning on Group Performance," *Journal of Applied Psychology* (77:5), pp. 682-693.

Williamson, O. E. 1981. "The Economics of Organization: The Transaction Cost Approach," *The American Journal of Sociology* (87:3), pp. 548-577.

Yamauchi, Y., Shinohara, T., and Ishida, T. 2000. "Collaboration with Lean Media: How Open-Source Software Succeeds," in *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, Philadelphia, Pennsylvania, PA, December 2-6, pp. 329-338.

Yates, J., and Orlikowski, W. J. 1992. "Genres of Organizational Communication: A Structurational Approach to Studying Communication and Media," *The Academy of Management Review* (17:2), pp. 299-326.

Yin, R. 1994. *Case Study Research: Design and Methods* (2md ed.) Newbury Park, CA: Sage Publications.

Yoo, Y., and Alavi, M. 2004. "Emergent Leadership in Virtual Teams: What Do Emergent Leaders Do?," *Information and Organization* (14:1), pp. 27-58.

Zammuto, R. F., Griffith, T. L., Majchrzak, A., Dougherty, D. J., and Faraj, S. 2007. "Information Technology and the Changing Fabric of Organizations," *Organization Science* (18:5), pp. 749-762.

Zuboff, S. 1988. *In the Age of the Smart Machine: The Future of Work and Power*, New York: Basic Books.

About the Authors

James Howison is an assistant professor in the School of Information the University of Texas at Austin. He received his Ph.D. in Information Science and Technology from the Information School at Syracuse University in 2009 and his B. Economics (Social Sciences) at the University of Sydney in 1998. Prior to Austin, James was a post-doc in the School of Computer Science at Carnegie Mellon University. His research focuses on technology and collaboration, especially open source software development and software work in science.

Kevin Crowston is a Distinguished Professor of Information Science in the School of Information Studies at Syracuse University, currently serving as a program director at the United States National Science Foundation. He received his Ph.D. (1991) in Information Technologies from the Sloan School of Management, Massachusetts Institute of Technology. His research examines new ways of organizing made possible by the extensive use of information and communications technology. Specific research topics include the development practices of Free/Libre Open Source Software teams and work practices and technology support for citizen science research projects, both with NSF support. He is currently a coeditor-in-chief for the journal *Information, Technology & People*, vice chair of the International Federation for Information Processing (IFIP) Working Group 8.2 on Information Systems and Organizations, and division chair-elect for the Academy of Management Organizational Communications and Information Systems Division.

MIS Quarterly

RESEARCH ARTICLE

COLLABORATION THROUGH OPEN SUPERPOSITION: A THEORY OF THE OPEN SOURCE WAY

James Howison
School of Information, University of Texas at Austin, 1616 Guadalupe Avenue,
Austin, TX 78701 U.S.A. {jhowison@ischool.utexas.edu}

Kevin Crowston
School of Information Studies, Syracuse University, 343 Hinds Hall, Syracuse, NY 13244 U.S.A.
and National Science Foundation {crowston@syr.edu}

Appendix A

Coding Scheme for Actions, Inductively Developed

Code	Explanation and Example
Management Codes	
Management	Work done to organize other work. This includes planning, setting deadlines or announcing "phases" like code/string freezes, assigning or rejecting tasks. This includes re-structuring the infrastructure and declaring bugs fixed, or patches applied (closing trackers).
Assigning credit	Thanking people, adjusting the credits file, etc.
Review Codes	
Validation	Validating a coding technique, fix, or approach (before or while it is being done).
Review	Work done to review other work, including checking in code written by others. This includes work that rejects patches, etc.
Production Codes	
Core production	Work that directly contributes to the project's outcomes, either through working application code or through production of user interface elements (logos, etc.); e.g., implementing a feature (not necessarily a check in, since could be checked in on behalf of someone else).
Polishing	Smaller changes that polish core production contributions; e.g., typos, integrations, etc.
Documentation Codes	
Documentation	Work that documents the code, application or activities. Includes pointers across venues (e.g., in a bug tracker saying that a patch has been submitted).
Self-Planning	Work that documents one's own future activities (planning others' work is management work).
Supporting Codes	
Use information provision	Providing or seeking information about using the software; e.g., use cases, often RFEs and bug reports.
Code information provision	Providing or seeking suggestions about the code, including how to complete work (code examples or pseudo-code, if it compiles or is a patch against SVN then code production work). This includes a developer seeking more information from a peripheral member.
Testing	Testing application functionality. This includes requesting more information from users in bug reports.

Appendix B

Summary of Important Terms

Term	Definition
Layer	An outcome of work in the form of a patch that can be applied to an existing, functioning, artifact.
Motivationally independent layer	A layer which, when applied to an existing artifact, provides sufficient motivating payoff without relying on the future completion of other layers.
Solo work	The production of a layer through the programming work of a single person.
Co-work	The production of a layer through the programming work of two or more people, where the motivational payoff for each participant is dependent on the successful completion of all participants' work.
Superposition	The process of laying down motivationally independent layers (thus extending an existing, functioning artifact). Analogous to the geological process of the sedimentary deposition of rock strata on top of another.
Open superposition	A search process led by freely available artifact use resulting in the conceptualization, production and superposition of motivationally and sequentially appropriate layers.
Collaboration through open superposition	A theory of "the open source way." The result of open superposition is that many people have contributed to a functionally interdependent artifact, through the autonomous production of motivationally independent layers. Thus open superposition is a kind of collaboration, even if it did not involve any co-work.

Appendix C

Methodological Appendix

This appendix draws together our methodological approaches for a comprehensive understanding of our method, since our presentation divides our work into three sections. We begin with our general philosophical approach, then discuss our participant observation, our archival replication, and our theory development.

Philosophical Approach

The overall philosophical perspective of this study is one of pragmatism (Diesing 1992; Goldkuhl 2008; Goles and Hirschheim 2000). Rather than beginning with a theoretical problem to be explained, our study began by participating in practice (methodological pragmatism), aiming to first provide a theoretical explanation for why that practice works (referential pragmatism), and finally to demonstrate our theory's usefulness by using it to consider adaptations of this practice (functional pragmatism). These goals informed the overall arc of the research, from deep engagement in the practice leading to theorizing about how and why the practices of FLOSS development work and broadening to asking whether these practices would work in other circumstances. Pragmatism also informed the investigation because constant comparison between participation and the evolving FLOSS literature showed which parts of the lived experience could not be usefully explained by that literature and, therefore, demanded new theorizing. Our theorizing was completed through engagement with practitioners, attempting to use existing theories and our developing theories in productive discussions.

Overall Methodology

The task of this project was always theory development. We were guided by the work of Weick (1989), who describes theory development as *disciplined imagination* directed to *sensemaking* and subject to iterative selection criteria, especially the characteristics of *plausibility and*

interestingness that develop through this selection process (see below). The selection process itself is realized through *representations*, which themselves require the discipline of writing, familiar from methodologies of ethnography (e.g., van Maanen 1988). In our case, these representations took the form of internal memos, academic publications, and presentations to and conversations with FLOSS developers, through an ongoing active research program, toward an understanding of effective work practices in free and open source software.[1] Below we describe the methodology in more detail for each of the three components of our study, highlighting the ways in which our imagination was disciplined and shaped through iterative selection pressures toward the theory presented in this paper.

Participant Observation

Our case study of BibDesk was an intensive field case study conducted through participant observation, an ethnographic method. The first author was prepared for this through studying case and ethnographic methods in Information Systems (Harvey and Myers 1995; Myers 1999; Yin 1994) and by reading relevant ethnographies (Barley 1986; Knorr-Cetina 1999; Yates and Orlikowski 1992; Zuboff 1988).

Participant observation derives insight from reflection on embedded, longitudinal, lived experience (Lewis 1985; Myers 1999). The first author, therefore, very specifically chose a product that could be integrated into his lived experience as a graduate student (a bibliographic manager). This project spanned four years in the field (exceeding standard recommendations for ethnographic length; Yin 1994, pp. 10-11), cycling up and down for active work episodes, but continually maintaining daily contact with the project through use of the application, as well as subscriptions to mailing lists and bug trackers, in the manner of open source "natives." Contact was limited to online, despite recommendations from some descriptions of participant observation methodology that encourage both interviews and informal social interaction with participants (e.g., Orlikowski 1991). The first author attempted this once, seeking a meeting with the project founder during a trip to San Diego (having noted that he shared an interest in surfing); the negative reply e-mail made it clear that such a social approach was not part of the culture for the project.

Throughout this period, the first author maintained field notes and produced periodic thematic memos for discussion, as recommended by Myers (1999). These notes were frequently discussed with the second author (Walsham and Sahay 1999), as well as a writing group of doctoral colleagues, and through presentations at doctoral consortia. During the analysis phase, the online archives of the project both online and in the e-mail client of the first author assisted in mapping back from field notes to original experiences. For example, not reported in this paper but described in Howison (2009), the observation and experience of primarily individual work was confirmed by a systematic analysis of BibDesk's archival records into episodes, counting aspects such as the number of participants in each different role. This systematic reconstruction functioned as a check on the participant observer's memory (as well as functioning as a pilot study for the archival replication).

The understandings presented in this paper evolved in interaction with two different discourses, disciplining our imagination with selection pressures (Weick 1989). The first was the academic literature grappling with FLOSS. Prior to entering the field, we identified three sensitizing concepts from this literature, discussed in detail in the main body of the paper, and as recommended for qualitative research (Bowen 2006; Glaser 1978; Patton 2002). Then, through our ongoing active research program, we repeatedly explored aspects of the FLOSS phenomena, trying out explanations and incorporating ongoing FLOSS research through a program leading to a FLOSS review paper (Crowston et al. 2012). We sought explanatory cohesiveness, but particularly focused on challenging the literature with lived experience, seeking areas in which they did not correspond. A key guiding principle was to identify and record in memos for discussion the understandings that were surprising in that they were in conflict with the assumptions in the academic literature before that different understanding became commonplace (Myers 1999). An example is the surprising disconnect between the (bursty) temporal rhythm of the lived experience and the flattened time of reading archives for other FLOSS research projects. Another example was the contrast between the focus on interdependency (and its management) in the literature and the experience of aloneness in the conduct and observation of work. A specific example of this occurred when the first author read Yamauchi et al. (2000) and felt it reported an important and little discussed aspect of what he was experiencing (independence), but felt that the explanation (lean media) did not accord with his experience. A second example was developed in conversation with a former Debian project leader who was undertaking FLOSS research as part of his Ph.D. in software engineering, (Michlmayr 2004; Michlmayr and Hill 2003). We discussed the manner in which the volunteer context and the lack of ability to coerce participants was central to discussion about the organization of work among FLOSS participants but strangely de-emphasized in the academic virtual teams literature on FLOSS development.

Simultaneously, we actively engaged with FLOSS practitioners in venues relevant to them, including Apachecon (three times), O'Reilly Open Source (twice), FOOcamp, the Australian Open Source Developers Conference, and LinuxWorld Asia (including a three-day high-level open source leaders speaker's tour to the Taj Mahal that provided occasion for detailed discussion). In these presentations and conversations, we brought academic perspectives (others' and our own evolving ones) and observed the extent to which participants found these *useful* for understanding their own experiences and judged them *interesting* (Davis 1971; Weick 1989). Practitioners, we found, drew blanks when we

[1] http://floss.syr.edu.

discussed purely social or management-driven explanations (such as decision making or leadership), often immediately recognizing these more as features of their "day jobs" that contrasted with their experience in FLOSS work. Instead, they responded with enthusiasm and engagement to explanations that discussed the organization of task work and its motivational context. This repeated pragmatic selection pressure helped draw out the findings presented in our participant observation.

Archival Replication

The archival replication is part of the process of theory building, rather than an attempt to verify or test that theory. We argue for its usefulness within that process, highlighting ways in which our sensemaking and imagination were disciplined. The work was undertaken to build confidence that the processes observed in our case study did not derive only from the specifics of our single case study and thus proved sufficiently disciplined for useful theorizing.

The reconstruction of the work episodes from archival records relies on a method which Geiger and Ribes (2011) would later call "trace ethnography," which involves a process of inversion (Bowker and Star 2000) that connects back from digital trace records to lived experience. This process draws on an ethnographic understanding of the "sociotechnical infrastructure of documentary practices" (Geiger and Ribes 2011) that is then drawn on in interpreting the archival records. Thus, within the overall process of theory development, the ethnographic understandings from participant observation were important in the archival reconstruction work conducted by the first author, disciplined as described below. The categories employed in this analysis are somewhat idealized, but are grounded in how developers think about the world, shown by the manner in which they contribute task outcomes to the release notes.

The overall purpose of the archival analysis was narrative reconstruction, focusing particularly on sequence and focal actors (Pentland 1999). There are two analytic moves in the reconstruction: the identification of tasks and the classification of actions within those tasks. This process was disciplined in three ways. First, tasks were identified through FLOSS team members' own records of the outcomes of their work, through individual bullets added to the release notes over time, and CVS log messages; these source are emic and form the anchors for the task. Second, the codebase itself, as a well-structured artifact, provided discipline: the analyst was able to trace a source code change in context, comparing between versions and tracing paths of execution (see Figure C2). Third, the analysis sought narrative cohesion, relying on the necessary logic of sequence: that some actions must precede others (e.g., writing code must proceed its review) (see Pentland 1999). These three principles discipline the analysis, resulting in experimental splits and merges of tasks, seeking narrative coherence.

The identification of the actions as of different types was conducted by the first author, disciplined by both narrative cohesion and, although a formal coding reliability test was not performed, an outside colleague's review of a sample of the tasks with their coded actions, confirmed narrative cohesion. The key finding (of few tasks in which more than one programmer writes code) relates to the identity of those checking code into CVS as recorded by the system, providing the fourth source of discipline. The analyst merely linked CVS user names and e-mail addresses across identities, drawing on database fields and hints in the textual data (such as initials, see Figure C1; see also Geiger and Ribes 2011).

The figures convey the interpretative apparatus. Figure C1 shows the release notes, providing the key anchors for the recognition of tasks. Figure C2 shows the Sourceforge CVS diff browser that enables understanding the code changes in context, browsing linked files and following lines of code execution. Figure C3 shows the interpretative coding apparatus (using RDF expressed in turtle triples); note the memos (relevance_memo, task_memo and ca_memo, a code application memo). The structured database constructed in this manner was then queried in two ways, the first to output each task in its sequence for further interpretation (as in Table 2 in the main body of the paper), the second to output counts of event attributes (such as number of programmers or length of task) used to report quantitative results and create the graphs shown in this paper.

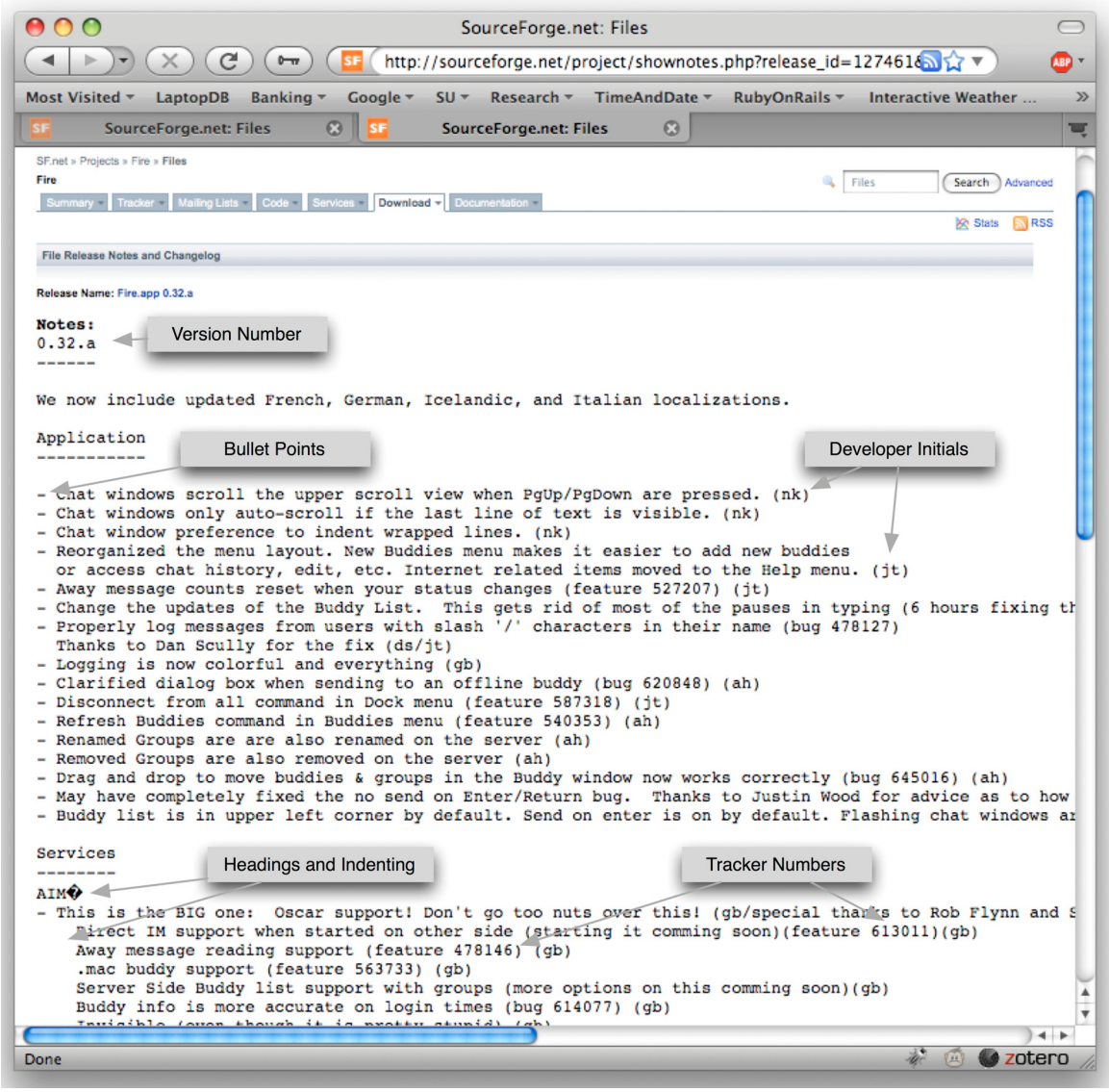

Figure C1. Release Notes Providing Task Outcomes as Anchors for Task Reconstruction

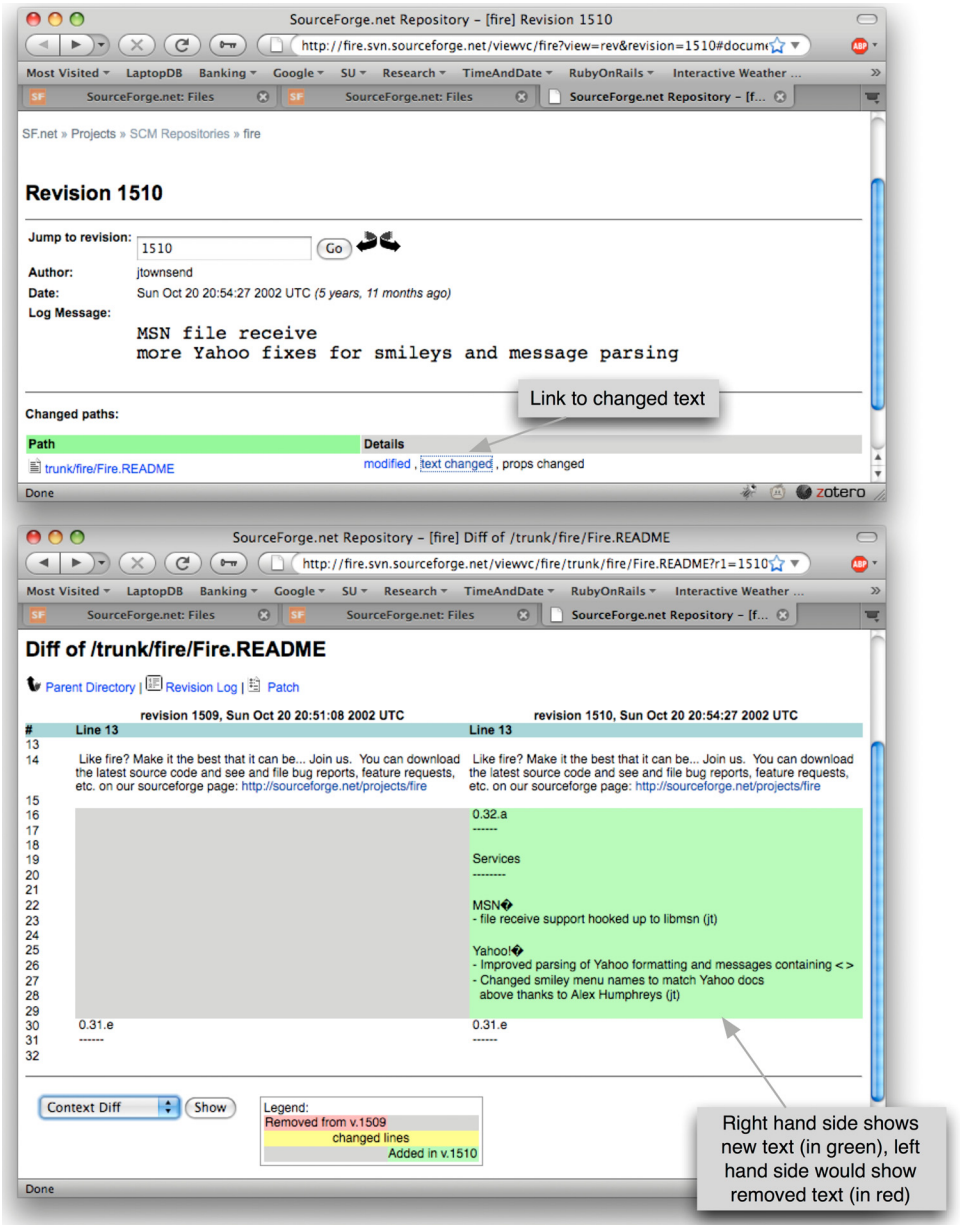

Figure C2. The Sourceforge CVS Browser, Facilitating Knowledge of Code in Context

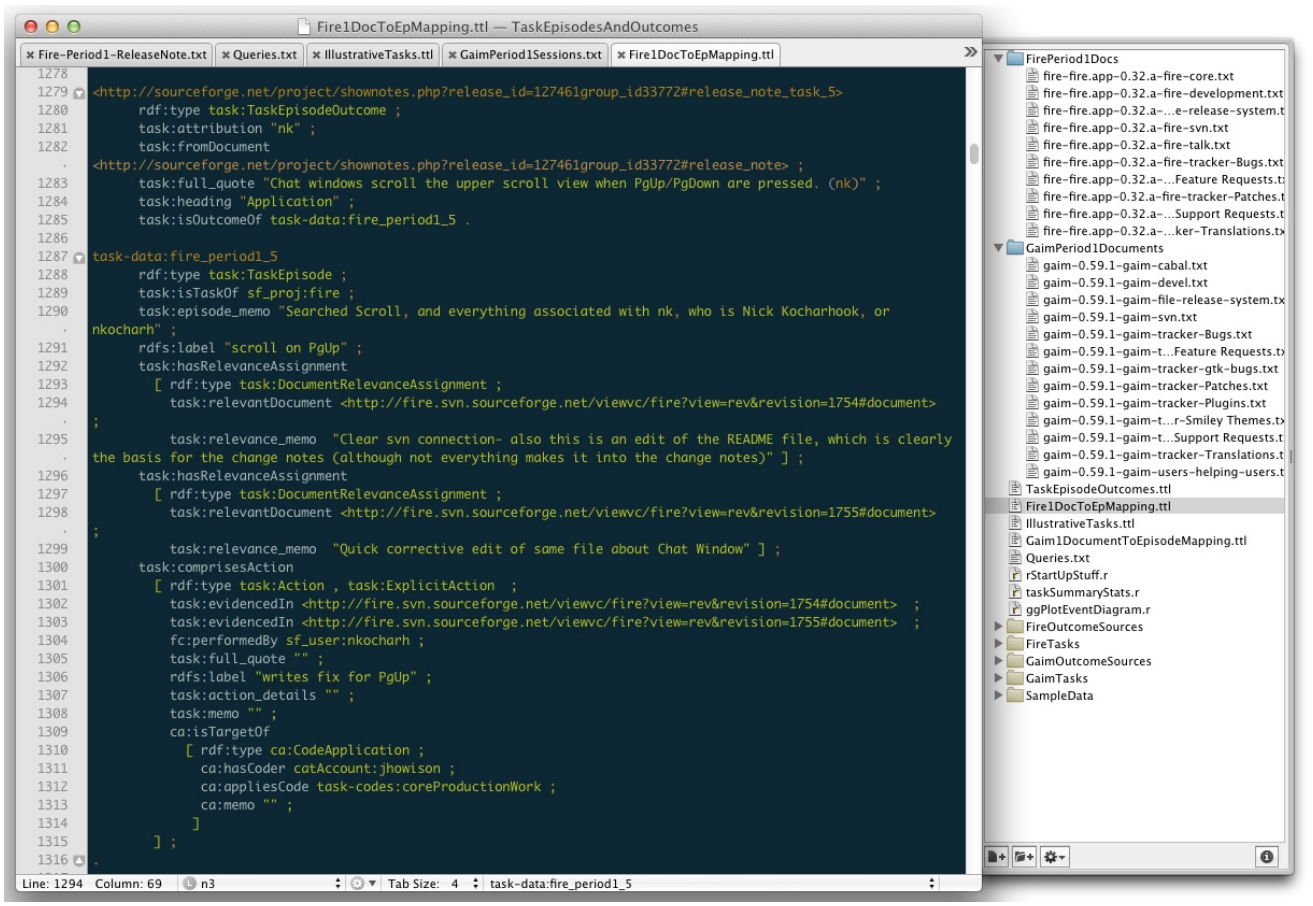

Figure C3. The Interpretative Apparatus, Producing Structured RDF Linking Archives, Memos, Codes, Actors, and Event Times

Theory Building

Weick's method for theory building relies on selection pressures toward "plausibility and interestingness," echoing the criterion of *plausibility* for ethnographic work discussed by Myers (1999) and Prasad (1997). Earlier, we described the iterative ongoing practice of working with both the academic literature and practitioners that provided the selection pressures. However, the selection characteristic of *interestingness* is potentially diffuse. Here we were guided primarily by Davis (1971), who argued that "interesting theories deny...weakly-held assumptions of their audience" (as quoted by Van de Ven 2007, p. 111). In our case, we deny two weakly held assumptions of the literature (rather, we hope we deny them and we hope they are only weakly held).

The first is an assumption of the virtual teams literature that FLOSS projects had found a way to coordinate and organize to accomplish complex work, despite the challenges of high interdependency of software work, working with volunteers, and working at distance. In contrast, we found that FLOSS projects simply chose not to undertake complex work but to defer it. Moreover, we found that deferring such complex work is key to the open source way of working.

The second assumption was that the core of the open source way of working was not closely bound to software work, and would thus relatively easily yield adaptable explanations (such as creation of trust, leadership, or decision-making techniques). In contrast, we found that software work (or rather, particular characteristics of software work that might conceivably be found elsewhere) were central to explaining the conundrum of how not doing complex work can ever lead to a collective complex artifact.

These aspects of interestingness, however, did not develop separately nor did they develop alone. Rather, we tried many explanations, both in conversation and in writing to both practitioners and academics, en route to these explanations. In particular, we returned to field notes and memos to choose and write the vignettes that illustrate our participant observation. In a second phase, through conversation with colleagues, reviewers, and multiple iterations, we confronted the need to face the question logically posed by the implications of the participant observation and archival reconstruction (How does mostly individual work result in a complex artifact?), clarifying the theory of collaboration through open superposition.

Finally, our pragmatic philosophical approach called for a demonstration of the usefulness of this theory. This we undertook through the analysis of the contingencies for the success of the theorized way of working and thereby casting light on the limits of the adaptability of the open source way. In these ways, our imagination was disciplined toward the theory presented in this paper.

References

Barley, S. R. 1986. "Technology as an Occasion for Structuring: Observations on CT Scanners and the Social Order of Radiology Departments," *Administrative Science Quarterly* (31:1), pp. 78-108.

Bowen, G. A. 2006. "Grounded Theory and Sensitizing Concepts," *International Journal of Qualitative Methods* (5:3), pp. 12-23.

Bowker, G. C., and Star, S. L. 2000. *Sorting Things Out: Classification and Its Consequences*, Cambridge, MA: MIT Press.

Crowston, K., Wei, K., Howison, J., and Wiggins, A. 2012. "Free (Libre) Open Source Software Development: What We Know and What We Do Not KNow," *ACM Computing Surveys* (44:2), Article 7.

Davis, M. S. 1971. "That's Interesting!," *Philosophy of Social Science* (1:4), pp. 309-344.

Diesing, P. 1992. *How Does Social Science Work? Reflections on Practice*, Pittsburgh PA: University of Pittsburgh Press.

Geiger, R. S., and Ribes, D. 2011. "Trace Ethnography: Following Coordination through Documentary Practices," in *Proceedings of the 44th Hawaii International Conference on System Sciences*, Los Alamitos, CA: IEEE Computer Society Press.

Glaser, B. G. 1978. *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*, Mill Valley, CA: Sociology Press.

Goldkuhl, G. 2008. "What Kind of Pragmatism in Information Systems Research?," in *Proceedings of AIS Special Interest Group on Pragmatist IS Research*, Paris, France, December 14, pp. 3-8.

Goles, T., and Hirschheim, R. 2000. "The Paradigm Is Dead, the Paradigm Is Dead…Long Live the Paradigm: The Legacy of Burrell and Morgan," *Omega* (28:3), pp. 249-268.

Harvey, L. J., and Myers, M. D. 1995. "Scholarship and Practice: The Contribution of Ethnographic Research Methods to Bridging the Gap," *Information Technology & People* (8:3), pp. 13-27.

Howison, J. 2009. "Alone Together: A Socio-Technical Theory of Notivation, Coordination and Collaboration Technologies in Organizing for Free and Open Source Software Development," unpublished Ph.D. dissertation, Syracuse University.

Knorr-Cetina, K. 1999. *Epistemic Communities*, Cambridge, MA: Harvard Education Press.

Lewis, I. M. 1985. *Social Anthropology in Perspective: The Relevance of Social Anthropology*, Cambridge, UK: Cambridge University Press.

Van Maanen, J. 1988. *Tales of the Field: On Writing Ethnography*, Chicago: University of Chicago Press.

Michlmayr, M. 2004. "Managing Volunteer Activity in Free Software Projects," in *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, Boston, June 27-July 2, pp. 39-50.

Michlmayr, M., and Hill, B. M. 2003. "Quality and the Reliance on Individuals in Free Software Projects," in *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Cork, Ireland, May 3-10, pp. 785-786.

Myers, M. 1999. "Investigating Information Systems with Ethnographic Research," *Communications of the Association for Information Systems* (2), Article 23.

Orlikowski, W. J. 1991. "Integrated Information Environment or Matrix of Control? The Contradictory Implications of Information Technology," *Accounting, Management and Information Technologies* (1:1), pp. 9-42.

Patton, M. Q. 2002. *Qualitative Research and Evaluation Methods*, London: Sage Publications.

Pentland, B. T. 1999. "Building Process Theory with Narrative: From Description to Explanation," *Academy of Management Review* (24:4), pp. 711-724.

Prasad, P. 1997. "Systems of Meaning: Ethnography as a Methodology for the Study of Information Technologies," in *Information Systems and Qualitative Research*, A. S. Lee, J. Liebenau, and J. I. DeGross (eds.), London: Chapman & Hall, pp. 101-118.

Van de Ven, A. 2007. *Engaged Scholarship: A Guide for Organizational and Social Research*, Oxford, UK: Oxford University Press.

Walsham, G., and Sahay, S. 1999. "GIS for District-Level Administration in India: Problems and Opportunities," *MIS Quarterly* (23:1), pp. 39-65.

Weick, K. E. 1989. "Theory Construction as Disciplined Imagination," *Academy of Management Review* (14:4), pp. 516-531.

Yamauchi, Y., Shinohara, T., and Ishida, T. 2000. "Collaboration with Lean Media: How Open-Source Software Succeeds," in *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, Philadelphia, Pennsylvania, PA, December 2-6, pp. 329-338.

Yates, J., and Orlikowski, W. J. 1992. "Genres of Organizational Communication: A Structurational Approach to Studying Communication and Media," *The Academy of Management Review* (17:2), pp. 299-326.

Yin, R. 1994. *Case Study Research: Design and Methods* (2md ed.) Newbury Park, CA: Sage Publications.

Zuboff, S. 1988. *In the Age of the Smart Machine: The Future of Work and Power*, New York: Basic Books.