Work Features to Support Stigmergic Coordination in Distributed Teams[*]

Kevin Crowston[1], James Howison[2], Francesco Bolici[3] and Carsten Østerlund[1]

[1] Syracuse University School of Information Studies
[2] University of Texas School of Information
[3] Università degli Studi di Cassino, Department of Economy and Law

To be presented at the Academy of Management Annual Meeting, August 2017

Abstract

When work products are shared via a computer system, members of distributed teams can see the work products produced by remote colleagues as easily as those from local colleagues. Drawing on coordination theory and work in computer-supported cooperative work (CSCW), we theorize that these work products can provide information to support team coordination, that is, that work can be coordinated through the outcome of the work itself, a mode of coordination analogous to the biological process of stigmergy. Based on studies of documents and work, we postulate three features of work products that enable them to support team coordination, namely having a clear genre, being visible and mobile, and being combinable. These claims are illustrated with examples drawn from free/libre open source software development teams. We conclude by discussing how the proposed theory might be empirically tested.

Keywords: coordination, distributed teams, stigmergy

[*] Partly supported by the National Science Foundation, award CHS 16-18444.

Work Features to Support Stigmergic Coordination in Distributed Teams

Introduction

We propose a theory that describes how coordination of work in distributed teams can be supported by the shared work products of the team. Distributed teams are groups of geographically dispersed individuals working together over time towards a common goal. Though distributed work has a long history (e.g., O'Leary, Orlikowski, & Yates, 2002), advances in information and communication technologies have been crucial enablers for recent developments of this organizational form (Ahuja, Carley, & Galletta, 1997: 165) and as a result, distributed teams have become popular (Martins, Gilson, & Maynard, 2004). Distributed teams seem particularly attractive for knowledge-based tasks such as software development because the work can be shared via the same systems used to support team interactions (Nejmeh, 1994; Scacchi, 1991).

While distributed teams have many potential benefits, distributed workers face many challenges. Watson-Manheim, Chudoba, and Crowston (2012) suggest that distributed work is characterized by numerous discontinuities, that is, a lack of coherence in some aspects of the work setting (e.g., organizational membership, business function, task, language or culture) that hinders members trying to make sense of the task and communication with others (van Fenema, 2002), or that produces unintended information filtering (de Souza, 1993) or misunderstandings (Armstrong & Cole, 2002). These interpretative difficulties, in turn, make it hard for team members to develop shared mental models of the developing project (Curtis, Walz, & Elam, 1990: 52; Espinosa et al., 2001). The presence of discontinuities seems likely to be particularly problematic for software developers (van Fenema, 2002), hence our initial interest in distributed software development. Studies of software development teams (Curtis, Krasner, & Iscoe, 1988;

Humphrey, 2000; Sawyer & Guinan, 1998; van Fenema, 2002; Walz, Elam, & Curtis, 1993) conclude that system development requires knowledge from many domains, which is thinly spread among different developers (Curtis et al., 1988). As a result, large projects require a high degree of knowledge integration and the coordinated efforts of developers (Brooks, 1975).

We focus in particular on coordination of distributed work, that is, how team members manage dependencies among tasks. Coordination has been a perennial topic in the study of teams, in empirical software engineering in particular. As Kalliamvakou put it, "To developers, collaboration is equivalent to managing independent contributions to the common whole" (Kalliamvakou, Damian, Singer, & German, 2014: 8). Coordination is clearly important for team effectiveness: for example, Cataldo and Herbsleb (2013) found that a failure to match coordination to coordination needs led to an increase in code defects.

Distributed work seems to present a particular challenge for coordination. More effort is required for interaction when participants are distant and unfamiliar with each others' work (Ocker & Fjermestad, 2000; Seaman & Basili, 1997). Considering software engineering again, the additional effort required for distributed work often translates into delays in software release compared to traditional face-to-face teams (Herbsleb, Mockus, Finholt, & Grinter, 2001; Mockus, Fielding, & Herbsleb, 2000). These problems are reflected in Conway's law (Conway, 1968), which states that the structure of a product mirrors the structure of the organization that creates it. Accordingly, it would be expected that splitting software development across a distributed team would make achieving an integrated product more difficult (Herbsleb & Grinter, 1999: 85).

Recent technology-supported team research has adopted a positive perspective, seeking to identify ways that technology use can improve team processes and outcomes, in contrast to a

problem-solving approach that seeks to identify problems to address (e.g., Carroll, Rosson, Farooq, & Xiao, 2009). And despite the numerous challenges, there are ways in which distributed work can in fact be better than face-to-face (Hollan & Stornetta, 1992). In particular, when work products are shared via a computer system, team participants can see the artefacts produced by remote colleagues as easily as those from local colleagues (Dabbish, Stuart, Tsay, & Herbsleb, 2014) and these artefacts can provide information to support team coordination. As we discuss below, coordination through artefacts (the stuff actually worked on, such as software or documents) is different than coordination through prior planning, roles or explicit discussion.

Setting: Open source software development teams

We illustrate the proposed theory in the context of Free/Libre Open Source Software (FLOSS) development teams, as examples of successful distributed teams with novel approaches to coordination. FLOSS is a broad term used to embrace software developed and released under an "open source" license allowing inspection, modification and redistribution of the software's source without charge. Key to our interest is the fact that most FLOSS software is developed by distributed teams, as developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their work primarily by means of computer-mediated communications (CMC) (Raymond, 1998; Wayner, 2000). Due to their distributed nature, these teams depend on processes that span traditional boundaries of place and ownership. The research literature on software engineering emphasizes the difficulties of distributed development, but the case of FLOSS presents an intriguing counter-example.

What is perhaps most surprising about FLOSS development processes is that developers appear often to eschew traditional project coordination mechanisms such as formal planning, system-level design, schedules, and defined development processes (Herbsleb & Grinter, 1999).

As well, many (though by no means all) FLOSS programmers contribute to projects as volunteers, and in most cases, without working for a common organization. Characterized by a globally distributed developer force and a rapid and reliable development process, FLOSS development teams somehow profit from the advantages and overcome the challenges of distributed work (Alho & Sulonen, 1998). Indeed, the subtext of many FLOSS studies is to understand how those work practices can be applied to other settings.

The growing research literature on FLOSS development has addressed a variety of questions. A complete review is beyond the scope of this paper: numerous review articles have surveyed research on FLOSS (e.g., Aksulu & Wade, 2010; Crowston, Wei, Howison, & Wiggins, 2012; Østerlie & Jaccheri, 2007; Rossi, 2004) and there exist Web sites with collections of articles . The most relevant work is the research on team work practices and in particular, research on team coordination, which will be reviewed below.

We have chosen this focus because studies of FLOSS teams (including our own) and of distributed teams more generally point to the possibility of a novel form of coordination (stigmergy, described below) that needs more research to fully understand but which could be useful if it could be generalized.

The "coordination paradox"— How can distributed teams coordinate without communicating?

In this section, we draw on our prior work on FLOSS coordination to describe the paradox that motivates this theorizing: the apparent ability of teams to coordinate with little or no explicit communication. This finding emerged from a study of how FLOSS developers coordinate their work (Bolici, Howison, & Crowston, 2016; Howison, 2009; Howison & Crowston, 2014). Somewhat unexpectedly, in the study we found little evidence of overt coordination of the development, i.e., FLOSS developers seemed to rarely communicate about

coding tasks. The lack of evidence was surprising considering the transparency of FLOSS projects: we expected to find direct, discursive communication in email or other discussion fora through which developers interact. But we found few examples. The lack of direct interaction around the work has echoes in our other research findings. For example, we found that developers mostly self-assign work rather than have it assigned to them (Crowston, Li, Wei, Eseryel, & Howison, 2007; Crowston, Wei, Li, Eseryel, & Howison, 2005) and often make decisions about code without explicitly evaluating options (Heckman et al., 2007; Heckman et al., 2006). Interestingly, when developers do discuss their work, they often refer directly to the software code.

One interpretation of these findings is that rather than coordinating explicitly, FLOSS developers rely instead on implicit coordination (Rico, Sánchez-Manzanares, Gil, & Gibson, 2008), e.g., by sharing well-developed mental models (Crowston & Kammerer, 1998; Espinosa, Slaughter, Kraut, & Herbsleb, 2007a, b) or shared understandings (Braunschweig & Seaman, 2013) of the task that allow them to determine what needs to be done without the need for explicit communication and coordination. However, while developers clearly have and rely on mental models of the task, it seems unlikely that these explain the paradox by themselves. First, the FLOSS development process is highly complex and ever changing. It seems impossible that developers can keep their mental models up to date given the ever-changing dependencies within the code and modifications made by numerous other developers. Second, the problem of coordination is exacerbated as the participation of developers waxes and wanes over time. FLOSS participants are not all experts but range from newcomers to experienced software engineers. For implicit coordination to be sufficient, we would need to explain how inexperienced participants develop mental models sufficiently robust enough to address the

coordination needs. In short, while implicit coordination is important, this mechanism is not sufficient by itself.

In this paper, we offer a complementary perspective on coordination. We focus here on the evidence presented above that on the infrequent occasions when they do interact, developers often refer to the code that they are collectively developing. We theorize that work can be coordinated through the outcome of the work itself, a mode of coordination analogous to the biological process of stigmergy (Grassé, 1959). Heylighen defines stigmergy thusly: "A process is stigmergic if the work…done by one agent provides a stimulus ('stigma') that entices other agents to continue the job" (Heylighen, 2007).

The question then is how work products can support coordination. From this perspective, we state a more specific question for our theorizing: What socio-technical affordances of shared work systems enable stigmergic coordination? By socio-technical affordances, we mean the features of the technology used and the practices around that technology. For example, the source code control systems commonly used by FLOSS developers provide notifications of code submissions; details of the implementation of this technical feature enable other developers to maintain awareness of the state of the code to support coordination. To interpret these change messages, developers likely need some level of technical skill and mental models of the code structure, another kind of affordance. They may also be accustomed to creating code in a way that is easier for others to interpret. The inherent nature of the coding task itself may create the need for specific kinds of coordination that are particularly amenable to stigmergy.

It is important to note that we are not arguing that stigmergic coordination completely replaces other forms of coordination. We rather see these different modes as complementary. Developers clearly still need to talk on occasion and there seems to be an important role for

shared mental models in being able to interpret the stigma to guide action. Kalliamvakou quoted FLOSS developers as saying "because the developers are speaking the same language it is easier," and "members are familiar with the idea of working this way and share the mentality behind it" (Kalliamvakou et al., 2014).

Literature review

In this section, we review related work that informs our theorizing.

Coordination theory

We first introduce the topic of coordination and present the fundamentals of coordination theory, the theoretical foundation for the proposed study. Coordination theory (1994) synthesizes the contributions proposed in different disciplines to develop a systemic approach to the study of coordination. In this perspective, studying coordination means analyzing the management of the dependencies that emerge among the tasks and components of a system. This definition of coordination is consistent with the large body of literature developed in the field of organization theory (e.g., Galbraith, 1973; Lawrence & Lorsch, 1967; Mintzberg, 1979; Pfeffer & Salancik, 1978; Thompson, 1967) that emphasizes the importance of interdependence in group work.

Malone and Crowston (1994) analyzed group action in terms of actors performing interdependent tasks to achieve some goal (i.e., in an organizational process (Crowston, 1997; Crowston & Osborn, 2003)). These tasks might require or create resources of various types. For example, in the case of software development, actors include the user and various members of the software development team. Tasks include translating aspects of a user's problem into system requirements and code, or bug reports into bug fixes. Resources include information about the users' problems and developers' time and effort.

Coordination theory defines coordination as "managing dependencies" and conceptualizes dependencies as arising between multiple tasks when the tasks use or create the same resources. Dependencies come in three kinds. First, flow dependencies match Thompson's sequential dependency (Thompson, 1967): one task creates a resource that a second uses. Flow dependencies create the need to manage the usability of the resource and the timing and location of its availability. Second, a fit dependency occurs when the output of two tasks must fit together in the creation of a common resource. Alternately, if the output of the two tasks is identical, there is potential synergy, as the duplicate work can be avoided. Finally, a shared resource dependency emerges among tasks that use a common resource (like Thompson's pooled dependency). Resources may also be directly interdependent, e.g., due to physical connections, in which case there can be dependencies between the tasks that use connected resources.

The key point in coordination theory is that dependencies create problems or potential synergies that require additional work to manage. Malone and Crowston (1994) called the tasks embodying this extra work coordination mechanisms. For example, if particular expertise is necessary to perform a given task (a task-actor dependency), then an actor with that expertise must be identified and the task assigned to him or her. There are often several coordination mechanisms that can be used to manage a given dependency. For example, mechanisms to manage the dependency between a task and an actor include (among others): (1) having a manager pick an appropriate subordinate to perform the task; (2) assigning the task to the first available actor, regardless of skill; (3) a labour market in which actors bid on tasks; and (4) self-assignment of tasks based on individual interest, as in FLOSS. To manage a usability subdependency, the resource might be tailored to the needs of the consumer (meaning that the consumer must provide that information to the producer) or a producer might produce to a

standard so the consumer knows what resource to expect. To manage shared use of a resource, tasks might take turns, first-come-first-served or be given a reserved time slot in which to use the resource.

All collaborative work requires some coordination and so coordination mechanisms may be useful in a wide variety of organizational settings. Conversely, organizations with similar goals achieved via the same set of tasks must manage the same dependencies, but may choose different coordination mechanisms, resulting in different processes. And mechanisms are themselves tasks that must be performed by some actors, so adding coordination mechanisms to a process may create additional dependences that must be managed in turn.

As an example of a coordination theory analysis, we can identify numerous dependencies in the software development process that need to be managed, implying the need for matching coordination mechanisms. Since developers work on the same codebase, there is a dependency between their work, requiring mechanisms for resource sharing to avoid conflicting changes. For example, Blincoe, Valetto, and Goggins (2012) used traces of developers' work on the same files to identify the potential need for two developers to coordinate. An important dependency is between a task (e.g., a bug report) needing to be done and someone to work on it, requiring mechanisms for task assignment (e.g., the bug might be assigned by a development manager to a developer to ensure that it is fixed by exactly one developer). Source code, can be managed proactively (developers check out the code they want to work on, preventing others from making conflicting changes) or optimistically (e.g., change made are checked for conflicts and any detected are resolved). And finally, the code itself has many interdependencies, e.g., a function that calls other functions or two that use shared data. Changing one piece of code can affect these

relationships (e.g., changing a function will require changing all places that call that function), requiring special attention when making changes.

We note that the coordination theory framework makes a distinction between the tasks and the mechanisms needed to coordinate the tasks. These two concepts are sometimes labeled "work" versus "articulation work" (Gerson & Star, 1986; Strauss, 1985). The conceptual split between work and coordination of work is also clear in the software engineering literature from Conway (1968) through Cataldo and Herbsleb (2008). The duality between work and coordination arises in part from an information-processing perspective on the work that assumes an input-process-output model of the work, making it natural to consider the tasks that create the output (connecting inputs to outputs) as the main part of the process and coordination mechanisms as separate from this work. Perhaps as a result, much of the focus of research on supporting coordination has addressed ways to improve explicit coordination. For example, an early CSCW system, the "Coordinator", sought to improve coordination by making communication more explicit about the coordination required (Flores, Graves, Hartfield, & Winograd, 1988; Winograd, 1987).

Stigmergic coordination

In contrast to the prior focus on explicit coordination, in this study we are interested in how the work itself can serve as guidance for coordination. For this analysis we draw on the biological process of stigmergy (Dipple, Raymond, & Docherty, 2014), defined as a process by which one individual affects the behaviour of others through changes in the shared environment. For example, ants follow scent trails to food found by other ants, thus assigning labour to the most promising sources. But the organized collective action emerges from the interaction of the individuals and the evolving environment, rather than from a shared plan.

While stigmergy was formulated to explain the behaviour of social insects following simple behavioural rules, it has also been invoked to explain classes of human behaviours: the formation of trails in a field as people follow paths initially laid down by others (similar to ant trails), or markets, as buyers and sellers interact through price signals (Parunak, 2006). For humans and intelligent systems, the signs and processing can be more sophisticated than is found for insects (Ricci, Viroli, Gardelli, & Oliva, 2007). For example, the shared environment can be a complex workspace including annotations. Tummolini and Castelfranchi (2007) developed a typology of different kinds of messages possible from signs, such as having the ability to do something, having done something or having a goal. Christensen (2007, 2013, 2014) discussed how architects and builders coordinate their tasks through "the material field of work" such as drawings, building on earlier work in CSCW focusing on coordination through the "field of work" including changes in shared databases (Schmidt & Simone, 1996).

Stigmergy has been suggested in particular as an interpretation of how FLOSS developers coordinate, what Kalliamvakou called a "code-centric collaboration" perspective (Kalliamvakou et al., 2014). FLOSS developers mostly work with the code that they are developing and source code control systems such as Git provide status about the state of the code and development. Dalle and David (2008) present a simulation of FLOSS developers allocating work based on information they get from the code base, providing evidence that stigmergy could be a viable approach to coordination in this setting. Stigmergy has also been argued as a mechanism in online work more generally. Elliot (2006) argued that "[c]ollaboration in large groups is dependent on stigmergy," with the specific example of authoring on Wikis.

Stigmergy can be readily interpreted in the coordination theory framework developed above. Malone and Crowston (1994) describe coordination mechanisms as relying on other

necessary group functions, including decision making, communications, and development of shared understandings and collective sense making (Britton, Wright, & Ball, 2000; Crowston & Kammerer, 1998). The stigmergic approach suggests that the "shared material" itself can be a communications medium, allowing coordination without recourse to separate coordinative mechanisms (Christensen, 2013). Christensen observed this type of coordination among architects, noting that their work is "partly coordinated directly through the material field of work…in addition to relying on second order coordinative efforts (at meetings, over the phone, in emails, in schedules, etc.), actors coordinate and integrate their cooperative efforts by acting directly on the physical traces of work previously accomplished by themselves or others" (Christensen, 2008).

The stigmergic perspective can be seen in the context of ongoing debates about the nature of socio-material structures for articulating the entwined nature of work and coordination (Østerlund, 2008b). Stigmergy resonates with newer theoretical perspectives on human activity that "recognise the importance of the environment", such as "activity theory, situated action, and distributed cognition" (Susi & Ziemke, 2001). A common element in these perspectives is the description of the role of artefacts in collaboration, i.e., what we are labelling as stigmergy. Stigmergy is also compatible with a structurational perspective on work. Structuration theory (Giddens, 1984) is a broad sociological theory that seeks to unite action and structure. The theory is premised on the duality of structures, meaning that the structural properties of a social system are both the means and the ends of the practices that constitute the social system. Jacob (2001) similarly argued that "stigmergic structures inform and control individual cognitive activities, they are themselves the residual products of successful problem-solving activities undertaken by the larger socio-cultural community".

Studies of stigmergy can also be informed by research on other concepts of long-standing interest in the field of computer-supported cooperative work (CSCW). First, there has been a stream of research in CSCW and elsewhere that demonstrates the importance of team member awareness for supporting collaborative work. Though they are not identical, there is clearly a close relationship between the two ideas about supporting collaboration. Christensen (2013) described actions a person might take to make a co-worker aware of an issue, and so distinguishes awareness from stigmergy, as "stigmergy does not entail making a distinction between the work and extra activities aimed solely at coordinating the work".

Similarly, in contrast to active awareness (one participant calling for the attention of another), Dourish and Bellotti (1992) argued for the importance of passive awareness mechanisms, which could be interpreted as supporting stigmergy. Other researchers have proposed awareness displays that allow a team member to develop an awareness of the actions of other team members. Carroll and colleagues (Carroll, Neale, Isenhour, Rosson, & McCrickard, 2003; Carroll, Rosson, Convertino, & Ganoe, 2006) examine in particular how awareness can support development of common ground, community of practice, social capital and human development in team. In this paper, we focus more narrowly on how awareness of work supports coordination.

A second related concept is system translucency (Erickson & Kellogg, 2000) or transparency (Colfer & Baldwin, 2010; Dabbish, Stuart, Tsay, & Herbsleb, 2013; Dabbish et al., 2014; Stuart, Dabbish, Kiesler, Kinnaird, & Kang, 2012), meaning visibility of details of organizational processes or functions. Consistent with our analysis of stigmergy, Stuart et al. (2012) analyze transparency as a form of information exchange or communication. They note that technology enables new forms of transparency, e.g., as in GitHub, a software development

site (Dabbish, Stuart, Tsay, & Herbsleb, 2012) that provides real-time updates on what other developers are doing. In other words, transparency is a system feature that might support awareness. System transparency might also support motivation to work, as participants respond to each other's visible work (Olson, Howison, & Carley, 2010). Researchers have noted similar problems with awareness and transparency, such as the potential for information overload from having to review too much information or that making too much visible may inhibit the willingness to share work (Bernstein, 2012; Dabbish et al., 2013).

As with stigmergy, system transparency provides information that can influence how people work. Dabbish et al. (2014) note specifically that transparency is helpful for coordination. They list numerous uses of visibility information, such as including dependencies with other projects (Dabbish et al., 2013). They further note that being able to see something means "much less need for routine technical communication" (Dabbish et al., 2013), suggesting that transparency is substituting for explicit coordination. Research on visibility and transparency can clearly be quite informative for designing systems to support stigmergic coordination. However, this stream of research has not specifically focused on the socio-technical affordances that enable users to make sense of and to use the provided stigma to support coordination, which is the goal of the current theorizing. For example, given the large number of possible signs available, how do developers decide which to attend to?

A third related concept in the CSCW literature is provenance, i.e., the history of a piece of information. Rather than being explicitly and independently created, provenance of documents is built as the documents are changed, or recorded from interaction as the documents are used, i.e., it is a kind of stigma. Hill, Hollan, Wroblewski, and McCandless (1992) and Wexelblat and Maes (1999) pointed out that knowing how others have interacted with a piece of information

can be informative for future interactions with it. Similarly, knowing the history of a document's development is important in evaluating and knowing how to use it.

There are of course many, many collaborative systems designed to support groups and in particular to support coordination of group work (that is, for managing dependencies among group tasks). However, only a few systems have been explicitly aimed at supporting stigmergic coordination. Musil, Musil, and Biffl (2014) proposed the concept of a Stigmergic Information System (SIS) architecture metamodel, though their goal is to develop an architectural model that describes many different kinds of systems rather than to build a particular one. Most of the systems described as stigmergic appear so far to focus on simply providing access to the shared work, without specific attention to coordination of the work. For example, Zhang, Zhao, Jiang, and Jin (2015) described a system for allowing collective construction of a conceptual model. Secretan (2013) described the Picbreeder system in which users interact only via images to create new images.

Theory building

In this section, we build an initial theory of the socio-technical affordances of systems necessary to support stigmergic coordination, with particular attention to the relation between stigmergy and other forms of coordination. It is hypothesized that these characteristics of systems for sharing work will support coordination of the work, thus distinguishing a system for stigmergic coordination from systems for explicit coordination on the one hand and systems for simple information sharing on the other.

Documents

To theorize what affordances of work support coordination, we turn to the literature on documents and work (Østerlund, Sawyer, & Kazianus, 2010). Software code is a semiotic product recorded on a perennial substrate that is endowed with specific attributes intended to facilitate specific practices (Zacklad, 2006), thus making it a kind of document. Code differs from other kinds of documents by serving two audiences, one being a machine, the other software developers. However, we focus on the latter, describing properties of code that allow developers to share their work with colleagues, and to read, understand and respond to their intentions.

Scholars have described how documentation and other accounts of work play a central role in the coordination of work (Bowker & Star, 1994, 1999; Østerlund, 2007, 2008a, b; Smith, 2005; Suchman, 1993, 1995). These perspectives have long pointed to the double role of documents as both "models of" work and "models for" work. For the first, documents provide an account "of" reality as workers manipulate text and other symbolic structures so as to parallel them with reality. For example, developers may carefully document the code they have constructed to create a report of the work done. But documents also provide a basis from which people further manipulate the world. For example, developers' reports are not simply accounts of work completed: the reports guide ongoing work by prescribing what is left to be done or enabling collaborators to coordinate their work. Taking inspiration from Smith (2005) and Bakhtin (1986), we suggest that a work product is rarely completely original; it is always an answer (i.e., a response) to work that precedes it, and is therefore always conditioned by, and in turn qualifies, the prior work. What the programmer does when facing somebody's work is responsive and partially determined by what has been going on up until now. The reports are

thus accounts "for reality" as they provide a blueprint of the software taking shape. Documents in this way offer a double accountability: when documenting the coding of a software program, developers mold the account to the reality of the code on their computers and at the same time, mold their ongoing coding to the account.

Three further concepts from document studies stand out as helpful in articulating how work can serve as a model for work: genre, visibility and mobility, and combinability. We address these in turn.

Document, genre and genre systems

First, people can recognize a document as a model for possible action only because they have some background knowledge about the genre of that document, and thus the expectations associated with that type of communication (Østerlund, 2007). A genre is defined as typified action invoked in response to a recurrent situation (Yates & Orlikowski, 1992). For example, common document genres related to a conference research paper include submission, reviews and acceptance letter. Each has a characteristic form (e.g., a review form) and purpose. People engage genres to accomplish social actions in particular situations, which are characterized by a particular purpose, content, form, time, place and set of participants.

The same can be said about FLOSS work products. A developer engages in typified actions invoked in response to recurrent situations. They do so to accomplish something characterized by a particular purpose, material form, place, time and participants. By completing a piece of code and leaving it for a colleague to work on, a developer invokes a specific genre of work. The colleague will be able to pick up and work with the code because it invokes that genre and so comes with certain expectations. The first engineer might have created a scaffold of a module that simply outlines a structure. In so doing, his work product becomes a model for work

associated with specific elements and course of action. It might invoke a sequence of steps or routes to a conclusion. It might invoke certain categories or socio-material arrangements that must be used. In this way, a piece of completed work serves as a model for future work by drawing on its own genre, i.e., what are the expected outcomes, what materials and forms should be invoked at what places and times and by what types of participants.

Furthermore, documents related to work (and so we argue, the work itself) are often organized into what are called genre systems (Orlikowski & Yates, 1994), formalized sequences of documents of particular genres providing more or less standardized methods for recognizing what might be done and what does get done as legitimate work. We alluded above to the genres surrounding a conference paper. The process of publishing a journal paper similarly involves a sequence of documents of specified genres, adding to the above a editor's report, revised manuscript, summary of revisions, final submission, galley proof, copyright release and published paper.

A key point in the analysis of work in terms of genres is that for genres and genre systems to enable documents to function as models for work they must be part of the conventions of practice shared among members of particular communities. Genres are not naturally occurring. They are rather learned as part of membership of such communities: As new participants are socialized into communities, they gradually acquire a naturalized familiarity with the socio-material arrangements and prominent genres.

Turning back to FLOSS, source code provides genre expectations and thus serves as a "model for" work at two levels. First, the FLOSS development process includes many distinct and typified actions involved in response to a recurrent situation and expressed in a set of characteristic documents, including source code, bug reports, commit messages and so on. These

genres are associated with particular purposes, forms, content, times, places and participants for the related tasks. For example, the purpose of the code is to instruct the machine to perform an application, whereas bug reports are used to provide information to developers about observed problems with a program. Code is used nearly exclusively by developers, while bug reports are shared between users and developers. By looking at these work outputs, experienced FLOSS participants can tell which tasks are called for.

Second, the source code itself has a structure in which each component has more-or-less well-defined purposes associated with particular functionalities. There are genres of source code: It collectively has the purpose of providing instructions for the computer, but as well, each module of a program has its own particular purpose and so its own subgenre. For example, some modules may manage the interface, while others deal with interactions among particular data sources. Clarity of communicative purpose is of critical importance for developers; it should be clear which components are appropriate to modify to add some desired new functionality. In a well-structured program, the purpose of each module is clear—the subgenre is recognizable—and so the code is useable by others as a model for work. In poorly structured code, the purpose of particular module may be hard to determine or, in fact, muddled and unclear. This confusion may not directly affect the functionality of the program, but in these cases, the code does not constitute a genre. Future programmers cannot tell where to add new functionality because the current work outcomes do not make it clear how to add it without negatively interfering with existing functionality. Consistent with this view, developers reflecting on success factors for FLOSS argue that developing the right program structure, one that communicates well to other developers, is key to the success of a FLOSS project.

FLOSS development further provides expectations about sequential ordering of documents. The broad genres of software development seem to be arranged in an organized manner where one task typically follows another. For instance, a developer would first read a bug report, then change the code and commit, creating a source code control system commit message. Releases have their own sequence of documents, such as release note, packaged software, binary distributions and so on. In other words, the FLOSS infrastructure supports certain sequences of documents that suggest particular sequences of processes that participants learn as part of membership in the group.

Visibility and mobility

The second key feature of documents is their visibility and mobility. For a piece of work to serve as a model for future actions it must be visible and accessible to others in a shared work zone (Kellogg, Orlikowski, & Yates, 2006). Obvious as it may seem, making work visible is not a straightforward process. As discussed by Suchman (1995), some work may be more visible than other work; some work may cover up previous activity and render it invisible. For example, service work is notoriously hard to make visible: the better such work is done, the less visible it is to those who benefit from it. CSCW research on awareness and transparency also addresses these concerns. Understanding what elements of work are accessible and how its visibility may change over time is central to understanding how work may or may not serve as a model for future work. Similarly, for work to coordinate tasks beyond a physically-restricted space, it must become mobile (Latour, 1990), meaning that it is conveyed to a context in which others can encounter it.

Most obviously, the FLOSS development infrastructures support the mobility of work by being Web-based. Any FLOSS developer can download the source code from the source code

control system and have access to others' work as a basis on which they can build their own. As a result, software engineers can, in many situations, use others' work as a model for their own work because of their ubiquitous access to the server containing the code. Further, many systems provide a mechanism to push changes to locally other developers' workspaces, rather than having to wait for those others to seek them out. By being in multiple places, code can coordinate work in multiple settings.

In FLOSS, the source code control system also records a revision history: all changes made to each file in the system including what files are created or deleted by whom, when (the file's provenance). Many changes include short notes that can explain why a change was made (although many changes do not, apparently expecting the reader to examine the code directly). Such histories not only serve as 'models of' work but can also point forward by depicting the generally accepted work process. For a newcomer, such histories provide a window to how things are done, what tasks tend to follow what tasks and what is regarded as good and opposed to bad (i.e., reverted) work. Visibility of FLOSS work is promoted as well through cultural norms about development. A widely acknowledged culture norm in open source is to "check in early, and check in often." If people do not share their work often, they are not making it visible to other participants to build on. Large infrequent commits ("code bombs") increase the chances that there will be conflicts and make it harder for other developers to understand what a change does, again hampering visibility. Indeed, a frequent complaint about a code contribution is that it is too large for developers to easily understand.

Combinability

The third important characteristic of work is combinability. For work to be a model for future work, the work must be combinable and improvable in increments that can be

superimposed on existing work (Howison & Crowston, 2014; Latour, 1990). If a piece of work is done, nothing is left to do, hence Raymond's surprising injunction to "leave low hanging fruit" (Raymond, 1998). Most work tasks are layered and complex: New work contributions can be adjusted and added to existing outcomes. A piece of work might start out as an incomplete frame, a scaffold on which other parts get added in some organized sequence. Later, new functionality can be added to the existing structure. In this way, a program evolves from version to version through a process described as "superposition" (Howison & Crowston, 2014).

Combinability in FLOSS development is supported both by the source code control system infrastructure and cultural norms. First, there are strong cultural norms for providing "atomic commits," that is, developers are encouraged to address only one change or topic when committing new code, leading to many small commits (Arafat & Riehle, 2009). It is easier to combine code with a focused commit than with a commit that does multiple things and touches bits and pieces of dozens of files in the process. It is likewise easier to back out a focused commit if things should go wrong. Developers are also warned: "Don't break the build", which means that the main set of files in the source code control system should always compile and run. This practice ensures that any developer who downloads the code will be able to work with it, supporting the individual development described above.

Combinability is further supported by the source code control system infrastructure allowing participants to try out experiments on the code in a branch before committing it. Developers can execute and test ideas at any time without interfering with others; they can run the software with their proposed changes and obtain direct feedback about the combinability and thus success or failure of their changes. This approach allows them to iteratively enhance their understanding of the task and to modify their strategy for managing dependencies between the

existing system and what they are trying to accomplish. In this way, developers can interact with the code base as they would engage in a conversation by continuously receiving feedback on their output. Thus, developers can avoid a lot of communication with co-developers, since their active engagement with the artefact provides substantial insights; one has less need to ask another what their intentions were when one can experiment with the codebase.

Summary

In summary, from the literature we understand why coordination is important and a theoretical framework for examining stigmergic coordination in teams. Task design determines what dependencies exist in the work and the coordination mechanisms needed. The literature also provides a starting point for identifying the socio-technical affordances that enable team members to make use of stigmergic coordination, e.g., the role of genres of work and the visibility and combinability of the work contributions. Again, the nature of the tasks will determine the genres of work that provide information to support coordination. At an individual level, prior work suggests that motivation to cooperate will be important (Dabbish & Kraut, 2004).

Conclusion

In this paper, we have presented a theory that predicts the features of shared work that enable the work to serve as a resource for coordination in distributed teams. Further research is needed to test this theory. Several approaches seem feasible.

First, field studies could be done with members of distributed teams (e.g., FLOSS developers) to determine how they use shared work to coordinate their own contributions, thus testing the predictions of the theory. To maximize the chance of seeing stigmergic coordination in action, it would be best to focus on large, complex, active projects with high levels of

distribution and heterogeneity in participation. The need for coordination will be most pronounced in larger teams working on larger and more complicated systems. Communication challenges will be most pronounced in teams with distributed members that lack frequent opportunities to communicate. As well, more active projects will produce more code changes that increase the need for awareness of what other members are doing.

A key research activity in such studies will be to identify prominent genres of information used and their role in genre systems. To assess visibility and mobility, one can identify how work products are accessed, examine the gap between production and use and check whether subjects can say where a particular piece of work came from. To assess combinability, one can examine how work provided by others is incorporated into the subject's own work.

A second possible approach for a study is to examine developer email mailing lists for mentions of looking at source code as a basis for making decisions about coordination. This approach has the advantage of possibly examining multiple developers in multiple teams, though the effort involved in coding data may be a barrier.

In addition to studying FLOSS teams in which stigmergic coordination is believed to happen, it would be instructive to study other kinds of distributed teams in which the work products have some but not all of the hypothesized necessary features. For example, interviews could be done with a few Wikipedia editors to understand how the evolving articles themselves guide editors' work and what limitations these have in supporting coordination.

Finally, future research could adopt a design science approach, seeking to build a system that embodies the recommendations of the theory and testing its capacity to support stigmergic coordination. As an example, consider developers collaboratively creating systems analysis

diagrams (e.g., UML class diagrams). A system could affect how changes to shared diagrams are made visible. Developers using single-user drawing tools need to share entire diagrams, which makes it difficult for recipients to identify specific changes. When using an online shared drawing tool (e.g., gliffy or LucidChart), changes are visible line by line, perhaps making it hard to understand the intent of changes. It may be that grouping changes in semantically-meaningful chunks and providing notifications in a form like code check-in messages better supports coordination by making it easier to understand what has changed and why. We expect that prior work on activity displays (Carroll et al., 2003; Carroll et al., 2006; Dabbish & Kraut, 2008) and translucency (Erickson, 2003; Erickson, Halverson, Kellogg, Laff, & Wolf, 2002) will be informative in the design process.

Combinability of contributions is also hypothesized to be important, but this feature is not explicitly addressed in prior work. Making contributions to a diagram combinable may be more challenging. It will also be important to develop social affordances that support stigmergy. Implementing social features may require training for users, provision of other kinds of information (e.g., overviews to help build mental models of the structure of the work) or careful selection of the tasks to be supported.

In summary, an empirical study would provide evidence for (or possibly against) stigmergic coordination as a mode of coordination in distributed work. At present, the evidence for stigmergy is mostly indirect. The results would also illuminate the possibilities and limits on the transfer of coordination mechanisms from FLOSS development to other domains.

Even in its current state, the proposed theory suggests the socio-technical affordances that enable the use of stigmergy in FLOSS development and extends current CSCW research on transparency by providing a theoretical framing for its impact on coordination. Stigmergic

coordination could be useful in a broad range of settings where team members seek to coordinate work while reducing the overhead of explicit communications.

References

Ahuja, M. K., Carley, K., & Galletta, D. F. 1997. Individual performance in distributed design groups: An empirical study. In ***Proceedings of the SIGMIS Computer Personnel Research Conference***: 160–170. San Francisco, CA.

Aksulu, A., & Wade, M. R. 2010. A comprehensive review and synthesis of open source research. ***Journal of the Association for Information Systems***, 11(11): 576–656.

Alho, K., & Sulonen, R. 1998. ***Supporting virtual software projects on the Web***. Paper presented at the Workshop on Coordinating Distributed Software Development Projects, 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Palo Alto, CA, USA.

Arafat, O., & Riehle, D. 2009. The commit size distribution of open source software. In ***Proceedings of the Hawaii International Conference on System Sciences (HICSS-42)***: 1–8. Hawaii, US.

Armstrong, D. J., & Cole, P. 2002. Managing distance and differences in geographically distributed work groups. In P. Hinds, & S. Kiesler (Eds.), ***Distributed Work***: 167–186. Cambridge, MA: MIT Press.

Bakhtin, M. M. 1986. The problem of speech genres. In C. Emerson, & M. Holquist (Eds.), ***Speech Genres and Other Late Essays: M.M. Bakhtin***: 60–102. Austin: University of Texas Press.

Bernstein, E. S. 2012. The transparency paradox: A role for privacy in organizational learning and operational control. ***Administrative Science Quarterly***, 57(2): 181–216.

Blincoe, K., Valetto, G., & Goggins, S. 2012. Proximity: A measure to quantify the need for developers' coordination. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 1351–1360. New York, NY, USA. doi: 10.1145/2145204.2145406

Bolici, F., Howison, J., & Crowston, K. 2016. Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms. ***Cognitive Systems Research***, 38: 14–22. doi: 10.1016/j.cogsys.2015.12.003

Bowker, G. C., & Star, S. L. 1994. Knowledge and information in international information management: Problems of classification and coding. In L. Bud-Frierman (Ed.) ***Information Acumen: The Understanding and Use of Knowledge in Modern Business***: 187–213. London: Routledge.

Bowker, G. C., & Star, S. L. 1999. ***Sorting Things Out: Classification and Its Consequences***. Cambridge: MIT Press.

Braunschweig, B., & Seaman, C. 2013. ***An examination of shared understanding in free/libre open source project maintenance***. Paper presented at the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE).

Britton, L. C., Wright, M., & Ball, D. F. 2000. The use of co-ordination theory to improve service quality in executive search. ***Service Industries Journal***, 20(4): 85–102.

Brooks, F. P., Jr. 1975. ***The Mythical Man-month: Essays on Software Engineering***. Reading, MA: Addison-Wesley.

Carroll, J. M., Neale, D. C., Isenhour, P. L., Rosson, M. B., & McCrickard, D. S. 2003. Notification and awareness: Synchronizing task-oriented collaborative activity.

International Journal of Human-Computer Studies, 58(5): 605–632. doi: 10.1016/S1071-5819(03)00024-7

Carroll, J. M., Rosson, M. B., Convertino, G., & Ganoe, C. H. 2006. Awareness and teamwork in computer-supported collaborations. ***Interacting with Computers***, 18(1): 21–46. doi: 10.1016/j.intcom.2005.05.005

Carroll, J. M., Rosson, M. B., Farooq, U., & Xiao, L. 2009. Beyond being aware. ***Information and Organization***, 19(3): 162–185. doi: 10.1016/j.infoandorg.2009.04.004

Cataldo, M., & Herbsleb, J. D. 2008. Communication networks in geographically distributed software development. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 579–588. San Diego, CA, USA.

Cataldo, M., & Herbsleb, J. D. 2013. Coordination breakdowns and their impact on development productivity and software failures. ***IEEE Transactions on Software Engineering***, 39(3): 343–360. doi: 10.1109/TSE.2012.32

Christensen, L. R. 2007. Practices of stigmergy in architectural work. In ***Proceedings of the ACM Conference on Supporting Group Work (Group)***. Sanibel Island, FL.

Christensen, L. R. 2008. The logic of practices of stigmergy: Representational artifacts in architectural design. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 559–568. New York, NY, USA.

Christensen, L. R. 2013. Stigmergy in human practice: Coordination in construction work. ***Cognitive Systems Research***, 21: 40–51.

Christensen, L. R. 2014. Practices of stigmergy in the building process. ***Computer Supported Cooperative Work (CSCW)***, 23(1): 1–19. doi: 10.1007/s10606-012-9181-3

Colfer, L., & Baldwin, C. 2010. ***The Mirroring Hypothesis: Theory, Evidence and Exceptions***. Working Paper 10–058, Harvard Business School, Boston, MA.

Conway, M. E. 1968. How do committees invent. ***Datamation***, 14(4): 28–31.

Crowston, K. 1997. A coordination theory approach to organizational process design. ***Organization Science***, 8(2): 157–175.

Crowston, K., & Kammerer, E. 1998. Coordination and collective mind in software requirements development. ***IBM Systems Journal***, 37(2): 227–245.

Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., & Howison, J. 2007. Self-organization of teams for free/libre open source software development. ***Information and Software Technology***, 49(6): 564–575. doi: 10.1016/j.infsof.2007.02.004

Crowston, K., & Osborn, C. S. 2003. A coordination theory approach to process description and redesign. In T. W. Malone, K. Crowston, & G. Herman (Eds.), ***Organizing Business Knowledge: The MIT Process Handbook***. Cambridge, MA: MIT Press.

Crowston, K., Wei, K., Howison, J., & Wiggins, A. 2012. Free/libre Open Source Software development: What we know and what we do not know. ***ACM Computing Surveys***, 44(2): 7:1–7:35. doi: 10.1145/2089125.2089127

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

Curtis, B., Krasner, H., & Iscoe, N. 1988. A field study of the software design process for large systems. ***Communications of the ACM***, 31(11): 1268–1287.

Curtis, B., Walz, D., & Elam, J. J. 1990. ***Studying the process of software design teams***. Paper presented at the International Software Process Workshop On Experience With Software Process Models, Kennebunkport, Maine, United States.

Dabbish, L., & Kraut, R. 2008. Research note—Awareness displays and social motivation for coordinating communication. ***Information Systems Research***, 19(2): 221–238. doi: 10.1287/isre.1080.0175

Dabbish, L., & Kraut, R. E. 2004. Controlling interruptions: Awareness displays and social motivation for coordination. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 182–191. New York, NY, USA.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. 2012. Social coding in GitHub: Transparency and collaboration in an open software repository. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 1277–1286. New York, NY, USA. doi: 10.1145/2145204.2145396

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. 2013. Leveraging transparency. ***IEEE Software***, 30(1): 37–43. doi: 10.1109/MS.2012.172

Dabbish, L., Stuart, H. C., Tsay, J., & Herbsleb, J. D. 2014. ***Transparency and coordination in peer production***. arXiv preprint 1407.0377.

Dalle, J.-M., & David, P. A. 2008. ***Motivation and coordination in Libre software development: A stygmergic simulation perspective on large community-mode projects***. Paper presented at the DRUID-SCANCOR Conference, Stanford University.

de Souza, P. S. 1993. ***Asynchronous Organizations for Multi-Algorithm Problems***. Unpublished Doctoral Thesis, Carnegie-Mellon University.

Dipple, A., Raymond, K., & Docherty, M. 2014. General theory of stigmergy: Modelling stigma semantics. ***Cognitive Systems Research***, 31–32(0): 61–92. doi: 10.1016/j.cogsys.2014.02.002

Dourish, P., & Bellotti, V. 1992. Awareness and coordination in shared workspaces. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 107–114. Toronto, Ontario, Canada. doi: 10.1145/143457.143468

Elliot, M. 2006. Stigmergic collaboration: The evolution of group work. ***m/c journal***, 9(2).

Erickson, T. 2003. Designing visualizations of social activity: Six claims. In ***Extended Abstracts on Human Factors in Computing Systems***: 846–847. Ft. Lauderdale, Florida, USA: ACM. doi: 10.1145/765891.766027

Erickson, T., Halverson, C., Kellogg, W. A., Laff, M., & Wolf, T. 2002. Social translucence: Designing social infrastructures that make collective activity visible. ***Communications of the ACM***, 45(4): 40–44. doi: 10.1145/505248.505270

Erickson, T., & Kellogg, W. A. 2000. Social translucence: an approach to designing systems that support social processes. ***ACM Transactions on Computer-Human Interaction***, 7(1): 59–83. doi: 10.1145/344949.345004

Espinosa, J. A., Kraut, R. E., Lerch, J. F., Slaughter, S. A., Herbsleb, J. D., & Mockus, A. 2001. Shared mental models and coordination in large-scale, distributed software development. In ***Proceedings of the International Conference on Information Systems (ICIS)***: 513–518. New Orleans, LA.

Espinosa, J. A., Slaughter, S. A., Kraut, R., & Herbsleb, J. D. 2007a. Familiarity, complexity, and team performance in geographically distributed software development. ***Organization Science***, 18(4): 613–630.

Espinosa, J. A., Slaughter, S. A., Kraut, R., & Herbsleb, J. D. 2007b. Team knowledge and coordination in geographically distributed software development. ***Journal of Management Information Systems***, 24(1): 135–169.

Flores, F., Graves, M., Hartfield, B., & Winograd, T. 1988. Computer systems and the design of organizational interaction. ***ACM Transactions on Office Information Systems***, 6(2): 153–172.

Galbraith, J. R. 1973. ***Designing Complex Organizations***. Reading, MA: Addison-Wesley.

Gerson, E. M., & Star, S. L. 1986. Analyzing due process in the workplace. ***ACM Transactions on Office Information Systems***, 4(3): 257–270.

Giddens, A. 1984. ***The Constitution of Society: Outline of the Theory of Structuration***. Berkeley: University of California.

Grassé, P.-P. 1959. La reconstrution du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportament de termites constructeurs. ***Insectes Sociaux***, 6(1): 41–80. doi: 10.1007/BF02223791

Heckman, R., Crowston, K., Eseryel, U. Y., Howison, J., Allen, E., & Li, Q. 2007. Emergent decision-making practices In Free/Libre Open Source Software (FLOSS) development teams. In ***Proceedings of the IFIP WG 2.13 Working Conference on Open Source Systems***. Limerick, Ireland.

Heckman, R., Crowston, K., Li, Q., Allen, E. E., Eseryel, Y., Howison, J., & Wei, K. 2006. Emergent decision-making practices in technology-supported self-organizing distributed teams. In ***Proceedings of the International Conference on Information Systems (ICIS)***. Milwaukee, WI, 10–13 Dec.

Herbsleb, J. D., & Grinter, R. E. 1999. Splitting the organization and integrating the code: Conway's law revisited. In ***Proceedings of the International Conference on Software Engineering (ICSE)***: 85–95. Los Angeles, CA.

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

Heylighen, F. 2007. Why is open access development so successful? Stigmergic organization and the economics of information. In B. Lutterbeck, M. Bärwolff, & R. A. Gehring (Eds.), ***Open Source Jahrbuch 2007***. Berlin: Lehmanns Media.

Hill, W. C., Hollan, J. D., Wroblewski, D., & McCandless, T. 1992. Edit wear and read wear. In ***Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)***: 3–9. New York, NY, USA.

Hollan, J., & Stornetta, S. 1992. Beyond being there. In ***Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)***. Monterey, CA. doi: 10.1145/142750.142769

Howison, J. 2009. ***Alone Together: A Socio-Technical Theory of Motivation, Coordination and Collaboration Technologies in Organizing for Free and Open Source Software Development***. Unpublished Doctoral Dissertation, Syracuse University, Syracuse, NY.

Howison, J., & Crowston, K. 2014. Collaboration through open superposition: A theory of the open source way. ***MIS Quarterly***, 38(1): 29–50.

Humphrey, W. S. 2000. ***Introduction to Team Software Process***: Addison-Wesley.

Jacob, E. K. 2001. The everyday world of work: Two approaches to the investigation of classification in context. ***Journal of Documentation***, 57(1): 76–99.

Kalliamvakou, E., Damian, D., Singer, L., & German, D. M. 2014. ***The code-centric collaboration perspective: Evidence from github***, Technical Report DCS-352-IR, University of Victoria.

Kellogg, K. C., Orlikowski, W. J., & Yates, J. 2006. Life in the trading zone: Structuring coordination across boundaries in postbureaucratic organizations. ***Organization Science***, 17(1): 22–44.

Latour, B. 1990. Visualisation and cognition: Drawing things together. In M. Lynch, & S. Woolgar (Eds.), ***Representation in Scientific Practice***. Cambridge: MIT Press.

Lawrence, P. R., & Lorsch, J. W. 1967. ***Organization and Environment***. Boston, MA: Harvard Business School.

Malone, T. W., & Crowston, K. 1994. The interdisciplinary study of coordination. ***Computing Surveys***, 26(1): 87–119.

Martins, L. L., Gilson, L. L., & Maynard, M. T. 2004. Virtual teams: What do we know and where do we go from here? ***Journal of Management***, 30(6): 805–835. doi: 10.1016/j.jm.2004.05.002

Mintzberg, H. 1979. ***The Structuring of Organizations***. Englewood Cliffs, NJ: Prentice-Hall.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. 2000. A case study of Open Source Software development: The Apache server. In ***Proceedings of the International Conference on Software Engineering (ICSE)***: 11 pages.

Musil, J., Musil, A., & Biffl, S. 2014. Towards a coordination-centric architecture metamodel for social web applications. In ***Proceedings of the European Conference on Software Architecture (ECSA 2014)***: 106–113. Vienna, Austria.

Nejmeh, B. A. 1994. Internet: A strategic tool for the software enterprise. ***Communications of the ACM***, 37(11): 23–27.

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

Ocker, R. J., & Fjermestad, J. 2000. High versus low performing virtual design teams: A preliminary analysis of communication. In ***Proceedings of the Hawai'i International Conference on System Sciences (HICSS-33)***: 10 pages.

Olson, J. F., Howison, J., & Carley, K. M. 2010. Paying attention to each other in visible work communities: Modeling bursty systems of multiple activity streams. In ***Proceedings of the International Conference on Social Computing (SocialCom)***: 276–281. doi: 10.1109/SocialCom.2010.46

Orlikowski, W. J., & Yates, J. 1994. Genre repertoire: The structuring of communicative practices in organizations. ***Administrative Science Quarterly***, 33: 541–574.

Østerlie, T., & Jaccheri, L. 2007. ***A Critical Review of Software Engineering Research on Open Source Software Development***. Paper presented at the AIS SIGSAND European Symposium on Systems Analysis and Design, Gdansk, Poland.

Østerlund, C. 2007. Genre combinations: A window into dynamic communication practices. ***Journal of Management Information Systems***, 23(4): 81–108.

Østerlund, C. 2008a. Documents in place: Demarcating places for collaboration in healthcare settings. ***Computer Supported Cooperative Work (CSCW)***, 17(2–3): 195–225.

Østerlund, C. 2008b. The materiality of communicative practice: The boundaries and objects of an emergency room genre. ***Scandinavian Journal of Information Systems***, 20(1): 7–40.

Østerlund, C., Sawyer, S., & Kazianus, E. 2010. Documenting work: A methodological window into coordination in action. In ***Proceedings of the Conference of the European Group for Organizational Studies (EGOS)***.

Parunak, H. V. 2006. A survey of environments and mechanisms for human-human stigmergy. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), ***Environments for Multi-Agent Systems II***, Vol. 3830: 163–186. doi: 10.1007/11678809_10

Pfeffer, J., & Salancik, G. R. 1978. ***The External Control of Organizations: A Resource Dependency Perspective***. New York: Harper & Row.

Raymond, E. S. 1998. The cathedral and the bazaar. ***First Monday***, 3(3).

Ricci, A., Viroli, A. O. M., Gardelli, L., & Oliva, E. 2007. Cognitive stigmergy: Towards a framework based on agents and artifacts. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), ***Environments for Multi-Agent Systems III***, Vol. 4389: 124–140: Springer. doi: 10.1007/978-3-540-71103-2_7

Rico, R., Sánchez-Manzanares, M., Gil, F., & Gibson, C. 2008. Team implicit coordination processes: A team knowledge–based approach. ***Academy of Management Review***, 33(1): 163–184.

Rossi, M. A. 2004. Decoding the free/open source puzzle: A survey of theoretical and empirical contributions. In J. Bitzer, & P. Schroder (Eds.), ***The Economics of Open Source Software Development: Analyzing Motivation, Organization, Innovation and Competition in the Open Source Software Revolution***: 15–55: Elsevier Press.

Sawyer, S., & Guinan, P. J. 1998. Software development: Processes and performance. ***IBM Systems Journal***, 37(4): 552–568.

Scacchi, W. 1991. The software infrastructure for a distributed software factory. ***Software Engineering Journal***, 6(5): 355–369.

Schmidt, K., & Simone, C. 1996. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. ***Computer Supported Cooperative Work: The Journal of Collaborative Computing***, 5: 155–200.

Seaman, C. B., & Basili, V. R. 1997. Communication and organization in software development: An empirical study. ***IBM Systems Journal*** 36(4): 550–563. doi: 10.1147/sj.364.0550

Secretan, J. 2013. Stigmergic dimensions of online creative interaction. ***Cognitive Systems Research***, 21: 65–74.

Smith, D. E. 2005. ***Institutional Ethnography: A Sociology for People***. Oxford: AltaMira Press.

Squire, M., & Crowston, K. 2015. ***FLOSShub***. http://flosshub.org/.

Strauss, A. 1985. Work and the division of labor. ***The Sociological Quarterly***, 26(1): 1–19.

Stuart, H. C., Dabbish, L., Kiesler, S., Kinnaird, P., & Kang, R. 2012. Social transparency in networked information exchange: A theoretical framework. In ***Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)***: 451–460.

Suchman, L. A. 1993. Technologies of accountability: Of lizards and aeroplanes. In G. Button (Ed.) ***Technology in Working Order***. London: Routledge.

Suchman, L. A. 1995. Making work visible. ***Communications of the ACM***, 38(9): 56–65.

Susi, T., & Ziemke, T. 2001. Social cognition, artefacts, and stigmergy: A comparative analysis of theoretical frameworks for the understanding of artefact-mediated collaborative activity. ***Cognitive Systems Research***, 2(4): 273–290.

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

Tummolini, L., & Castelfranchi, C. 2007. Trace signals: The meanings of stigmergy. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), ***Environments for multi-agent systems III***: 141–156: Springer. doi: 10.1007/978-3-540-71103-2_8

van Fenema, P. C. 2002. ***Coordination and control of globally distributed software projects***. Unpublished Doctoral Dissertation, Erasmus University, Rotterdam, The Netherlands.

Walz, D. B., Elam, J. J., & Curtis, B. 1993. Inside a software design team: Knowledge acquisition, sharing, and integration. ***Communications of the ACM***, 36(10): 63–77.

Watson-Manheim, M. B., Chudoba, K. M., & Crowston, K. 2012. Perceived discontinuities and constructed continuities in virtual work. ***Information Systems Journal***, 22(1): 29–52. doi: 10.1111/j.1365-2575.2011.00371.x

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

Wexelblat, A., & Maes, P. 1999. Footprints: History-rich tools for information foraging. In ***Proceedings of the SIGCHI Conference on Human Factors in Computing Systems***: 270–277. New York, NY, USA.

Winograd, T. 1987. A language/action perspective on the design of cooperative work. ***Human–Computer Interaction***, 3: 3–30.

Yates, J., & Orlikowski, W. J. 1992. Genres of organizational communication: A structurational approach to studying communication and media. ***Academy of Management Review***, 17(2): 299–327.

Zacklad, M. 2006. Documentarisation processes in documents for action (DofA): The status of annotations and associated cooperation technologies. ***Computer Supported Cooperative Work (CSCW)***, 15(2-3): 205–228.

Zhang, W., Zhao, H., Jiang, Y., & Jin, Z. 2015. Stigmergy-based construction of internetware artifacts. ***IEEE Software***, 32(1): 58–66. doi: 10.1109/ms.2014.133