Decision-Making Processes in Community-based Free/Libre Open Source Software Development Teams: The Impact of Shared Work Products

Kangning Wei, Shandong University (kwei.su@gmail.com)
Kevin Crowston, Syracuse University (crowston@syr.edu)
U. Yeliz Eseryel, University of Groningen (u.y.eseryel@rug.nl)
Robert Heckman, Syracuse University (rheckman@syr.edu)

Draft of 12 December 2016

Decision-Making Processes in Community-based Free/Libre Open Source Software Development Teams: The Impact of Shared Work Products

Abstract

Prior research on decision-making in Free/Libre Open Source Software development (FLOSS) teams has examined the distribution of decision-making power but the specific process by which decisions are made is still largely unexplored. Understanding these processes is practically and theoretically interesting. The lack formal organizational structures to guide practices and reliance on asynchronous communication might be expected to make decision making problematic but despite these challenges many FLOSS teams are quite effective.

This study adopted a process-based perspective to analyze decision-making in five community-based FLOSS teams developing two kinds of software (IM clients and ERP systems). Five different decision-making processes were identified, indicating FLOSS teams use multiple processes when making decisions. We explored the impact of three contextual factors on the complexity of different processes: decision type, trigger type and project type. The results suggested that strategic decisions adopt more complex processes than tactical decisions do. An unexpected finding was that teams developing more or less complex software seem to adopt similar practices. This research provides theoretical insights into decision-making in community-based FLOSS teams and has practical implications for FLOSS practitioners.

Keywords: Decision making; Decision process; Free/Libre Open source software development; Decision Types; Software Complexity

1 INTRODUCTION

As an essential component of team behaviour (Guzzo and Salas 1995), decision making has been extensively studied. Understanding decision-making processes in teams is important because the effectiveness of these processes can have a large impact on overall team performance (Hackman 1990). Decision making is of particular interest in information systems research because these processes are often supported and influenced by advanced information and communication technologies (ICT) such as expert systems and decision support systems

(Huber 1990, Shaikh and Karjaluoto 2015).

Decision-making researchers have argued for research that focuses on the process by which a team arrives at a team decision rather than simply relating inputs to decision outcomes (Guzzo and Salas 1995, Poole and Baldwin 1996). In examining the process of decision making, Poole and Roth (1989) argue that a decision is not a discrete, independent event but rather involves a series of activities and choices. Accordingly, they suggest that team-level decision making should be analyzed using a process approach, that is, as an interrelated sequence of events that occur over time leading to an organizational outcome of interest (Robey et al. 2000), namely a decision. A process approach allows us to analyze how the individuals interact and carry out various steps in the process, while the process as a whole has a team-level outcome (Crowston 2000).

In this research, we investigate decision-making processes in a specific type of team: community-based Free/Libre Open Source Software (FLOSS) development teams[1]. More specifically, to understand decision-making dynamics, we examine interpersonal interaction patterns, as interaction is the essence of team decision making (Poole and Baldwin 1996). FLOSS teams can include participants with varying levels of engagement, from core developers who write the majority of the code, to co-developers who contribute occasional bug fixes, to active users who contribute bug reports and feature requests (Amrit and Van Hillegersberg 2010, Barcellini et al. 2014). We view the developers and active users as forming a team: the team members have a shared identity, have a shared goal of developing a product to satisfy their user base and are interdependent in their tasks. As an open innovation model enabled by the use of information technologies (e.g., von Hippel and von Krogh 2003, Eseryel 2014), community-based FLOSS development (referred to simply as FLOSS throughout this paper)

[1] FLOSS development can be organized in different ways, reflecting different kinds of participants and different business models built around the software. In this paper, we investigate FLOSS in its original form, community-based teams, which are started by a small group and developed independently of any organization or company (Boldyreff et al. 2009), excluding from our study projects with significant company or foundation participation that might affect the decision processes.

has attracted great interest among researchers who seek to understand this novel model of openness, often with an interest in transferring the model to other settings (e.g., Economides and Katsamakas 2006, Oh and Jeon 2007).

Our interest in understanding the decision-making process in FLOSS teams was motivated by three distinct characteristics of the FLOSS context that we expected would pose barriers to effective decision-making, requiring novel decision-making processes.

First, FLOSS teams are generally virtual, as developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily by means of ICT (Crowston et al. 2012). The extensive use of ICT changes the way members can interact and so how they make decisions (Kiesler and Sproull 1992). A lack of shared context and numerous discontinuities in communication faced by virtual team members can hamper decision making (Watson-Manheim et al. 2002). In FLOSS teams, not only is the communication technologically mediated, but so too is the work itself. Yet, how the technology mediation of work may change the decision-making process has not yet been investigated.

Second, given the distributed nature of the work, asynchronous decision-making processes are required in order to enable participants from all around the world to contribute despite time-zone differences. Decision-making processes adopted by the team members are asynchronous in nature. FLOSS teams in particular rely on information technologies, such as team email lists, discussion fora, websites, bug trackers and source code repositories, for communication, coordination and discussion of alternatives, what Barcellini et al. (2014) term discussion spaces. A further complication is that prior research has suggested that information technologies can affect the decision-making mechanisms (Kiesler and Sproull, 1992). Asynchronous communications make it difficult for the participants to catch various clues in communication that are available in synchronous media such as voice tone, speed, and body language. The lack of such cues creates barriers to decision-making process since sense-making and understanding become more difficult for the participants. Furthermore, our understanding of which contextual

factors influence decision-making processes that take place on various information technologies is highly limited.

Third, unlike organizational teams, community-based FLOSS teams are generally self-organizing, meaning that leaders are not externally appointed. Indications of ranks or roles are materialized through interaction rather than external cues, meaning that there is no hierarchical source of decision authority[2]. However, in this setting, decision-making is even more important, as decisions provide the basis for organizing. FLOSS members, similar to most members in engineering settings, value technical contributions over all else and are said to eschew positional power.

In FLOSS development teams, the knowledge base for the decision and the decision-making actions are widely accessible and those interested can contribute with their opinions and knowledge with minimal barriers, which might lead to different levels of complexity in different decision-making processes. In comparison to traditional organizations, it has been found that more people in FLOSS development can share power and be involved in team's activities (Crowston et al. 2012). The more participants and more discussions are involved in the decision-making processes, the more knowledge is accessible and transparent to many others, the more complex becomes the decision-making process.

Examination of how decision-making processes are adapted in the face of these characteristics will extend our understanding of team decision making. Furthermore, understanding how the technological systems that support and constrain virtual work affect decision-making processes should be informative for many kinds of knowledge work, which is increasingly virtual. At a more specific level, knowledge of FLOSS decision-making process can be informative for organizations or firms collaborating with FLOSS teams (Santos et al. 2013). As organization's engagement in FLOSS development is not passive (Colombo et al. 2014),

[2] This characteristic might not apply to projects in which company or foundation participants shape adopted practices, hence our focus on community-based projects in this study.

understanding the decision-making processes in FLOSS is critical for organizations hoping to extract the most values from their interactions with these communities.

While decision making has been recognized as an important function in FLOSS teams (Crowston et al. 2012), prior FLOSS decision-making studies examine decision making primarily by examining governance, leadership and authority. For example, studies have examined the distribution of decision-making power (e.g., German 2003, Fitzgerald 2006) and found that participants nearer to the core have greater control and discretionary decision-making authority compared to those further from the core. Research has further categorized different governance mechanisms and approaches to leadership in different FLOSS teams. A connection has been observed between hierarchical governance structure and centralization of decision-making processes (Gacek and Arief 2004). The centralized decision-making process in Linux Kernel (Moon and Sproull 2000) has been characterized as a benevolent dictatorship (Raymond 1998). In contrast, the relatively non-hierarchical GNOME team has a decentralized decision-making process involving task-forces (German 2003). Finally, roles and decision-making structures have been observed to be dynamic (Raymond 2001, Robles 2004, Nelson et al. 2006). Fitzgerald (2006) suggests that early in the life of a team, a small subset will control decision making, but as the software grows, more developers will get involved.

To sum up, the FLOSS research on decision-making has to date examined the general governance of FLOSS, specifically who has decision-making power. However, there is a lack of empirical research on the specific details of the FLOSS decision-making process. This gap is illustrated in the lack of coverage of the topic in published FLOSS review and research framework articles (Nelson et al. 2006, Niederman et al. 2006, von Krogh and von Hippel 2006, Scacchi 2007, Aksulu and Wade 2010, Crowston et al. 2012). To fill this gap, we address the following general research question in this paper: How are decision-making processes structured in community-based FLOSS development teams? More specifically, based on the contingency model of decision-making processes (which will be discussed in detail in section 2),

we explore the following two specific questions:

1. *What decision-making processes emerge in community-based FLOSS development teams?*
2. *How do contextual factors influence the complexity of decision-making processes in community-based FLOSS development teams?*

To answer these questions, we first analyze decision episodes from five FLOSS teams to identify distinct decision-making patterns. Then using statistical analysis, we investigate whether different contextual factors influence the complexity of the decision-making processes used.

2 THEORETICAL BACKGROUND

To investigate decision-making process, it is important to clarify what constitutes a "team decision" in FLOSS settings. We define FLOSS team decisions as those explicit and implicit consensus decisions (Kerr and Tindale 2004) that bind the team and the external users of the software as a whole to a future course of action, e.g., decisions about which bugs to fix and how or which features to add, as well as more strategic decisions related to social, organizational, strategic and legal aspects of development.

Explicit consensus refers to a case where all or most of the team members participate in the decision process and explicitly state their agreement with the decision (e.g., by voting). Implicit consensus refers to occasions where one or more members make the decisions in a public forum, meaning that all team members can observe the decision due to the openness provided by the ICT, but where there is no explicitly expressed agreement or disagreement from others. The idea of implicit consensus reflects the fact that in FLOSS teams, communication and work rely on open broadcast media, and so are transparent to all. Thus, any teamwork that is shared and not rejected by others has been implicitly agreed to by the rest of the team. Of course, apparent implicit consensus may also be a result of non-participation in the process, but repeated non-participants have essentially ceased to be team members, meaning that implicit decisions still reflect a consensus of the active team participants.

In this section, we first review prior theories on team decision-making processes as a basis for identifying decision-making process in FLOSS teams. Then building on prior research, we identify three contextual factors and discuss how they might impact the complexity of different decision processes.

2.1 Phasal Theories of Team Decision-Making Processes

A number of frameworks have been proposed to describe the phases of team decision-making processes. A phase is defined as "a period of coherent activity that serves some decision-related function, such as problem definition, orientation, solution development, or socio-emotional expression" (Poole and Baldwin 1996). Early studies proposed normative models to describe how decisions are made in a unitary sequence of decision phases (Poole and Roth 1989), which suggest that teams follow a systematic logic to reach decisions (Miller 2008).

However, Poole and his colleagues suggest that the normative models are not adequate to capture the dynamic nature of decision-making sequences, and propose another class of phase models, multiple-sequence models (Poole 1983, Poole and Roth 1989). In these models, teams might also follow "more complex processes in which phases repeat themselves and groups cycle back to previously completed activities as they discover or encounter problems. Also possible are shorter, degenerate sequences containing only part of the complement of unitary sequence phases" (Poole and Baldwin 1996). Based on a study of 47 team decisions, Poole and Roth (1989) identified 11 different decision processes that fell into three main groups: unitary, complex and solution-centered sequences. The sequences in these processes typically emerge spontaneously during the decision making, rather than being planned by the team ahead of time.

Multiple-sequence models of decision making are advantageous because they not only capture the complexity of the decision-making process that may vary due to factors such as task structure (Poole 1983), but also provide a systematic approach to studying the dynamic

decision-making processes (Mintzberg et al. 1976, Poole and Roth 1989). Further, multiple sequence models provide guidance for practitioners to adapt to changing demands (Poole 1983, Poole and Baldwin 1996) by providing a framework for structuring analyses of decision processes, terminology and a basis for comparison between diverse processes. We therefore adopted this approach in this paper.

As a starting point for our analyses, we use the extant literature on sequence models and the studies which identify decision-making process phases based on team communications analyses (Mintzberg et al. 1976, Poole and Baldwin 1996). Specifically, we adapted the Decision Functions Coding System (DFCS) developed by Poole and Roth (1989) to the FLOSS context to identify different decision-making processes in FLOSS context. The details of our adaptation are discussed below in section 4.2.

2.2 Contextual Factors Affecting the Level of Complexity of Decision-Making Processes

In investigating decision-making processes, it is important to control for factors that may influence the nature of the decision-making process. Under different circumstances, teams might follow different paths as they make decisions. Poole and Roth (1989) propose a contingency model to describe how team members structure their decision-making processes differently subject to the contextual factors of the decision situation. They found decision characteristics to be effective in predicting the sequences of decision paths and the complexity of the paths. Campbell (1988) argues that "any objective process characteristic that implies an increase in information load, information diversity or rate of information change can be considered as a contributor to process complexity" (p. 43). Decision task types and the underlying decision process complexity have been studied in the decision-making literature (Payne 1976, Speier et al. 2003). In this study, we consider three task-related factors that lead to different kinds of decision to be made, namely decision type, trigger type and project type, and discuss how these factors affect decision process complexity.

2.2.1 The impact of decision types on the complexity of decision-making processes

There are in general two levels of decision-making in software development activities: tactical decisions, the day-to-day activities that maintain efficient operations of developing and testing software functionality, and strategic decisions, the decisions concerned with the long-time health of a project (Drury et al. 2012). Tactical decisions are related to routine decisions about the primary work of the team, that is, software development, e.g., acceptance of bug fixes, additions of new features or product enhancements through a change in software. Such decisions are defined as team decisions because the source code is the shared product of the team and changes to the code thus affect all developers. Strategic decisions are decisions about the strategic direction for the project, social, organizational and legal issues, or alliances and partnerships.

Wood (1986) analyzes decisions as having three essential components: products (entities produced by behaviors that can be observed independently of the behaviors that produce them), required acts (behavior(s) required to create a defined product) and information cues (facts that can be processed to make conscious judgments). These components interact to affect the use of appropriate decision processes (Wood 1986). Tactical decisions and strategic decisions are different in all of these three elements.

First, while tactical decisions result in a tangible technical product of software code, strategic decisions result in non-technical and non-tangible product of consensus reaching on an aspect of the team or its process.

Second, the required acts for tactical decisions include development-related acts such as identification of potential technical solutions, evaluating different solutions and identifying the best solution. These acts are relatively well-structured as they are based on specific routines of software development procedures, such as design and testing. Required acts for strategic decisions include defining the issue, identifying relevant information and trying to build

consensus, which are less structured and thus are expected to require more discussion and participation, therefore resulting in a higher level of complexity of decision-making processes.

Third, tactical decisions require technical cues, whereas strategic decisions require social or process-related cues. As well, as noted earlier, while team members communicate and make decisions through mailing lists or discussion fora about both tactical and strategic decisions, tactical decisions can take advantage of an additional communications channel, that is, the software code itself. Software code is an active ICT artifact in the sense that interacting with the software (e.g., by testing it) gives developers direct feedback and provides explication of knowledge and insights without direct discussion with other team members (Bolici et al. 2015).

The communicative value of software has been noted for coordination of software development. The function has been argued as an analogy to the biological process of stigmergy, the way that ants coordinate their behaviors by changing their shared environment by leaving chemical traces (Crowston et al. 2011). Similarly, studies have argued that coordination of software development activities can take place through the work itself, i.e., through the shared software source code (Robles et al. 2005, Crowston et al. 2011). Bolici et al. (2015) argue that software code constitutes a shared artifact that enables the communication of implicit knowledge of other developers through interaction with the software code. Other researchers have noted that when individuals engage with the software code, they learn from how the software code is written, therefore transfer this tacit knowledge and create new tacit knowledge (Morner & von Krogh, 2009; Eseryel, 2014).

Software code may be a particularly useful kind of communication channel for decision-making processes, as it is on the one hand a form of explicit knowledge, and on the other, conveys tacit knowledge through the way it is structured and built (Haefliger et al. 2008, Morner and von Krogh 2009, Eseryel 2014). As a result, decision-making processes about the software do not need to be spelled out as much, i.e., need not be as complex. In FLOSS development in particular, each developer can access the software code (i.e., the artifact of the work) at any

time to inspect the changes made by the other developers (Bolici et al. 2015). As a result, developers making changes do not have to explain in detail what they have done: if others are curious, they can examine the code themselves. Based on this argument, we suggest that tactical decisions should have less need for explicit communication, since part of the necessary knowledge transfer happens when team members analyze and decode the knowledge embedded in the software itself (Eseryel, 2014).

Contrariwise, strategic decisions usually faces greater uncertainty as the information required in such decisions is usually incomplete. The decision-making process may extend over a considerable period of time and require participation and discussions from a broader team of participants (Moe et al. 2012). Some FLOSS projects even have established a formal way to deal with these decisions so that a sufficient number of participants are involved. For example, GNOME project has committees and task forces composed of volunteers to complete important strategic tasks (German 2003).

Moreover, for strategic decisions, mailing lists or discussion fora might be the only channels through which team members share knowledge with each other as these decisions do not benefit from communication via the software code. Therefore, the team members need to go through the more complex process of explicating all their knowledge and the knowledge of other relevant parties, such as asking questions to each other to make sure they understand each other's messages.

In summary, because of the differences in products, necessary acts and information cues, we expect that the decision-making processes for tactical decisions will be less complex than those for strategic decisions. Hence, we hypothesize:

H1: *Decision-making processes for tactical decisions will be less complex than those for strategic decisions.*

2.2.2 The impact of decision triggers on the complexity of decision-making process

Decision triggers are the stimuli that evoke decision processes. Different triggers can be expected to provoke different decision processes. A common distinction is between decision processes triggered from problems vs. opportunities (Mintzberg 1973, Smart and Vertinsky 1977). Opportunity triggers are typically ideas, whereas problem triggers may vary in multiple stimuli (Mintzberg et al. 1976, Nutt 1984). In FLOSS development, identification of bugs, or individuals' emails asking for help in resolving a problem they ran into are examples of problems that might trigger a decision process (Annabi et al. 2008), while new ideas, such as suggesting new features to be included, are examples of opportunities.

In tactical decisions, problem triggers usually refer to the written software code having some embedded problem. As we argued above, the existing software code can serves as a stigmergic mechanism, helping communicate and transfer knowledge and thus reduces the need for complex communication and coordination in the decision-making process. In contrast, opportunity-triggered tactical decisions usually talk about future plans for software development such as possibilities of adding new features. Since these issues are not yet realized in the source code, the affordances of stigmergic communication are not available. Therefore, more complex communication and coordination is necessary to convey knowledge to others and make decisions, which increases the complexity of the decision-making processes. Hence, we hypothesize:

H2a: *Problem-triggered tactical decisions will have simpler decision processes than the opportunity-triggered tactical decisions.*

In the case of strategic decisions, neither problem-triggered nor opportunity-triggered decisions are directly related to the product (i.e., the software source code) development or realized in the source code. These decisions have a wide range of options, limited only by the team members' imagination on the ideal future state of the affairs. Therefore, regardless of the

trigger type, we expect these decisions to be similarly complex. Therefore, we hypothesize:

H2b: *Problem-triggered strategic decisions will have equally complex decision processes as opportunity-triggered strategic decisions.*

2.2.3 The impact of project complexity on the complexity of decision-making processes

The third contextual factor we consider is the type of project, which refers to the type of the software a FLOSS team tries to develop. Specifically, we expect decision-making processes, for either tactical or strategic decisions, to be affected by the complexity of the software being developed. Collaboration in complicated tasks is dependent on the partners' knowledge and prior experience within the project teams. Research has found that task complexity affects team interaction and software development processes in general (Straus and McGrath 1994, Mennecke et al. 2000, Clarke and O'Connor 2012). In software development, the complexities of decisions vary greatly depending on the characteristics of the developed software itself, such as the size and structure of the software (Espinosa et al. 2007). Complex decisions require more acts and contain more information cues than simple decisions, and these cues and acts can be highly interdependent (Wood 1986). Further, researchers have argued that, with the high complexity of an IT project, it is more critical to have developers and users involved in the development process (Park and Lee 2014).

In addition to internal complexity of the software, Espinosa et al. (2007) suggest that coordination challenges imposed by external factors increase the complexity of software development, which in turn affects the decision-making process. For example, to develop accounting modules in enterprise resource planning (ERP) systems, system developers must consider legal requirements (e.g., the tax calculation rules for each country and region) and general business process requirements (e.g., generally accepted accounting principles). These imposed external factors increase the dependency of the decision makers on informational cues about such external factors.

These internal and external factors contribute to an increased level of complexity of the decision-making process in FLOSS projects that develop complex software. Following the argument above, we hypothesize:

H3: *FLOSS projects that develop simpler products will have simpler decision processes than the projects that develop more complex products.*

3 RESEARCH METHOD

We turn now to the design of a study to address our research questions and test our three hypotheses. Given the exploratory nature of our research and the complexity of the research question, we designed a two-phase multi-level (team level and decision-process level), multi-method (qualitative and quantitative) study. In phase 1, we collected 300 decision episodes from five FLOSS projects and content analyzed the episodes to identify distinct decision-making processes, which answers specific research question 1. Phase 1 also provides data regarding the different decision processes, which is then used in phase 2 to test the proposed hypotheses. We note though that both phases of the project are based on prior theory. That is, unlike many QUAL-QUANT studies, the role of the first phase of this study is not to develop hypotheses to test in the second phase, but rather to convert qualitative data into quantitative data to support hypothesis testing.

We sought to choose projects that would provide a meaningful basis for comparison across the three contextual factors. As previously noted, FLOSS business models are diverse. To control unwanted systematic variance, we chose community-based projects (the focus of our study) that were roughly similar in age and in potential market size, and that were all at production/stable development status. Projects at this stage have relatively developed membership and sufficient team history to have established decision-making processes, yet the software code still has room for improvement, which enables us to observe rich team interaction processes. Acknowledging that the development tools used might structure the decision-making processes, we selected projects that were all hosted on SourceForge (www.sourceforge.net), a

FLOSS development site popular at the time of data collection that provides a consistent ICT infrastructure to developers.

Specific research question 1 was answered by examination of decision-making processes in the sample of projects to see if similar patterns of decision-making processes emerged across different projects that use the same tools. Testing the hypotheses required comparisons of decisions with different task types (i.e., tactical versus strategic decisions) and different trigger types (i.e., problems vs. opportunities), and comparison of decision processes adopted by FLOSS teams that differ in the type of software they develop. For the later, we planned a theoretical comparison of projects developing low and high complexity software. Specifically, we selected projects that developed Instant Messenger (IM) clients and Enterprise Resource Planning (ERP) systems.

ERP software development is known to be complex (Ryan 1999). Tasks in ERP projects are expected to be more complex than those in IM projects since ERP systems have high software code interdependencies and many external constraints such as accounting rules and legal reporting requirements. ERP software developers also need to consider how the software can be engineered to fit the needs of diverse companies. In contrast, IM clients can be more generic, and thus simple, in serving the needs of many. These factors are expected to make decision-making processes in ERP projects different from those in IM projects.

We initially chose 3 cases for each project type: Gaim (currently known as Pidgin), aMSN and Fire from IM projects, and Compiere, WebERP and OFBiz (currently known as Apache OFBiz[3)] from ERP projects. However, during data analysis we came to realize that Compiere was not a community-based project like the others, since it was started by a company and now has both community and commercial aspects in its development. To avoid possible bias introduced by this project, we decided to remove this project from our study, resulting in 5 (3 IM

[3] At the time of the study, OFBiz was not under the Apache umbrella but was a community-based FLOSS project like the other selected projects.

and 2 ERP) projects in the final design.

In the following sections, we describe the research method in each phase and report the findings in detail. Specifically, section 4 describes the first phase of the research, including the description of the qualitative design to identify different decision-making processes and the corresponding results. Section 5 presents the second phase of the study, beginning with the description of the measures of the other constructs used for the study and ending with the results of the hypothesis testing.

4 PHASE 1: IDENTIFICATION OF THE PATTERNS OF DECISION-MAKING PROCESSES

4.1 Data and Unit of Analysis

To find evidence of decision-making processes, we analyzed the email discourse on the developers' email lists or fora, which are the primary communication venues for the members of these FLOSS teams. Data were obtained from the FLOSSmole website (http://flossmole.org/). Though we cannot completely rule out the possibility of off-list discussions occurring through other channels (e.g., IRC, IM, phone or face-to-face meeting), the teams used the email lists/fora as the main communication tool for development, meaning that such off-list discussions would be invisible to numerous developers as well as to us as researchers. Furthermore, our analysis of the mailing list interactions did not reveal references to such off-line discussions, suggesting that this data source provided a complete view of the decision-making process, at least for the decisions made here.

As our primary unit of coding and analysis, we selected the decision episode, defined as a sequence of messages that begins with a decision trigger that presents an opportunity or a problem that needs to be decided and that includes the required acts of issue discussion and which possibly ends with a decision announcement (Annabi et al. 2008). To give an example, a decision trigger may be a feature request or a report of a software bug. A decision announcement may be either a statement of the intention to do something or an actual

implementation of a fix. Note that some decision processes did not result in a decision that was announced to the group, while others had multiple announcements as the decision was revised. The messages in an episode capture the interactions among team members that constitute the process of making a particular decision from start to finish.

Decision episodes were identified from the continuous stream of available messages through an initial coding process done independently by two of the authors. We started the analysis by reading through the messages until we identified a message containing a decision trigger or announcement. Once we found a trigger or announcement, we identified the sequence of messages that embodied the team process for that decision. We observed that teams generally organize discussions in a thread, occasionally initiating new threads with the same or similar subject line. Therefore, we developed a decision episode by combining one or more discussion threads that used the same or a similar subject line as the initial message and that discussed the same main issue. Our explorative evaluation of the threads showed that any such follow-ups were typically posted within the following month, and in more extreme cases within 3 months. We therefore searched for messages on the same or similar content up to three months after the posting date of the last message on a thread. Since we were analyzing the messages retrospectively, we could collect all of the messages related to the decision over time.

The process of identifying messages to include in each episode proceeded iteratively, as the two researchers collected messages, shared the process they used with the research team, and revised their process as a result of feedback from the team. The pairwise inter-coder reliability reached 85% and 80% respectively on decision triggers and announcements. All differences between coders were reconciled through discussion to obtain the sample of episodes for analysis.

Sampling of decision episodes was stratified by time: we chose 20 episodes from the

beginning, middle and end periods of each project[4] based on a concern that the decision-making process might be different at different stages of the software development (e.g., initial collaboration vs. a more established team). However, χ^2 tests on the coded data (described below) showed no significant differences ($\chi^2 = 4.288$, df=4, p=0.368) in decision processes across the different time periods, so we combined all episodes for each project for our analysis.

The result of this initial coding process was a collection of 300 decision episodes, each a collection of messages with a trigger and decision announcements if any. The sample size was chosen to balance analysis feasibility with sufficient power for comparisons. With 60 episodes per project, we have reasonable power for comparison across projects while keeping the coding effort feasible.

4.2 Coding Scheme Development for Decision Processes

Once we had a sample of decision episodes, we content analyzed them by coding the segments of text that embodied the decision-making steps to identify decision-making process in each episode. The coding scheme was developed deductively in two steps. First as noted above, we adopted the Decision Functions Coding System (DFCS) developed by Poole and Roth (1989). This coding system uses as the primary unit of coding the "functional move", which is defined as "the function or purpose served by a particular segment of the conversational discourse" (Wittenbaum et al. 2004). Functional moves have been used extensively to understand the nature of interaction in both face-to-face and computer-mediated environments (Poole et al. 1985, Poole and Holmes 1995, Herring 1996). However, few studies have used functional move to analyze complex, asynchronous, text-based environments such as email, bulletin boards or threaded discussion fora. We used functional moves to identify the function of messages in each episode. Note that a single message might include zero, one or multiple

[4] For each project, the beginning and the ending periods were the first and last 20 decision episodes found as of the time of data collection (i.e., from the start of the project's on-line presence to the most recent period). The middle period for each project consisted of 20 episodes surrounding a major software release approximately halfway between the beginning and ending periods. We chose to sample around a release period because making a release is one of the key team decisions for a FLOSS project.

functional moves.

In DFCS, functional moves for decision making include steps for problem analysis and problem critique; orientation and process reflection; solution analysis, design, elaboration, evaluation and confirmation; and other conversational moves such as simple agreement. To use the DFCS for decision making, we first sorted the decision activities according to Mintzberg, et al.'s (1976) proposed decision-making process. The result is an "IDEA" framework with four overall phases, namely decision identification (I), development (D), evaluation (E) and announcement (A). Each phase includes one or more specific functional moves.

Second, the scheme was revised to adapt to the FLOSS setting. To adapt the scheme, we pilot coded a sample of 20 episodes and discussed how the scheme applied to the data. As a result of these discussions, we removed from the coding scheme the functional moves that seemed to not be applicable to the FLOSS context (such as "screening issues" and "authorizing decisions") and identified and added levels of detail that are unique to the FLOSS content that had not been seen in previous studies. Using the revised scheme, we then coded a further 20 episodes and discussed the results until no new patterns emerged. The details of the revision and the final revised coding scheme are given in Appendix 1.

According to this coding scheme, if the coders observe a perfectly rational decision-making process, the decision should go through all of the four phases represented by the following sequential activities:

(I) In the identification stage, the team members first identify an opportunity for decision-making (I-1), such as determining a need for a fix. The team members exchange information to understand the underlying problems (I-2).

(D) The development stage may start by discussing how such problems are generally resolved (D-1). Team members either look for existing solutions (D-2) or try to design a specific solution for the problem (D-3).

(E) At the evaluation stage, team members evaluate the options identified in the previous

stage, either by sharing their general evaluative opinions (E-1) or by testing the solutions and reporting the outcomes (E-2). Sometimes a team member initiates voting to determine the final solution or asks confirmation for a proposed solution (E-3).

(A) Finally, in the announcement stage, the final team decision on how the issue will be solved is presented to the group (A-1).

Figure 1 provides an example of how these functional moves were coded based on an example from the Gaim project. This processes went through all four phases of identification, development, evaluation and announcement consecutively, however making loops back twice from the evaluation stage to the previous development phase. While many dynamic decisions loop back almost at every stage, for simplicity, we chose to show an example where only two loop-backs happened.

Once we had a coding scheme established, two analysts independently coded the functional moves in the collected decision episodes, and then compared their results. The initial coding revealed about 80% agreement. Discrepancies were discussed until the analysts fully agreed on each code. After all disagreements were resolved, the coding was repeated until the analysts fully agreed on all coded segments. This iterative coding process took about one month.

A problem in analyzing process data is that at the most detailed level, processes can show great variability, making it hard to find theoretically meaningful patterns. To address this problem, we clustered the 300 coded decision episodes along the following two dimensions based on the sequences of moves represented in the episodes. The first dimension is the coverage, referring to the extent that theoretically-identified decision-making phases are observed in the public process. The second dimension is termed as cyclicity, i.e., whether the decision episodes progressed linearly through the phases as in a normative model or looped through phases repeatedly as suggested by researchers such as Mintzberg et al. (1976). From here on, we refer to these two categories as linear and iterative decision-making processes respectively.

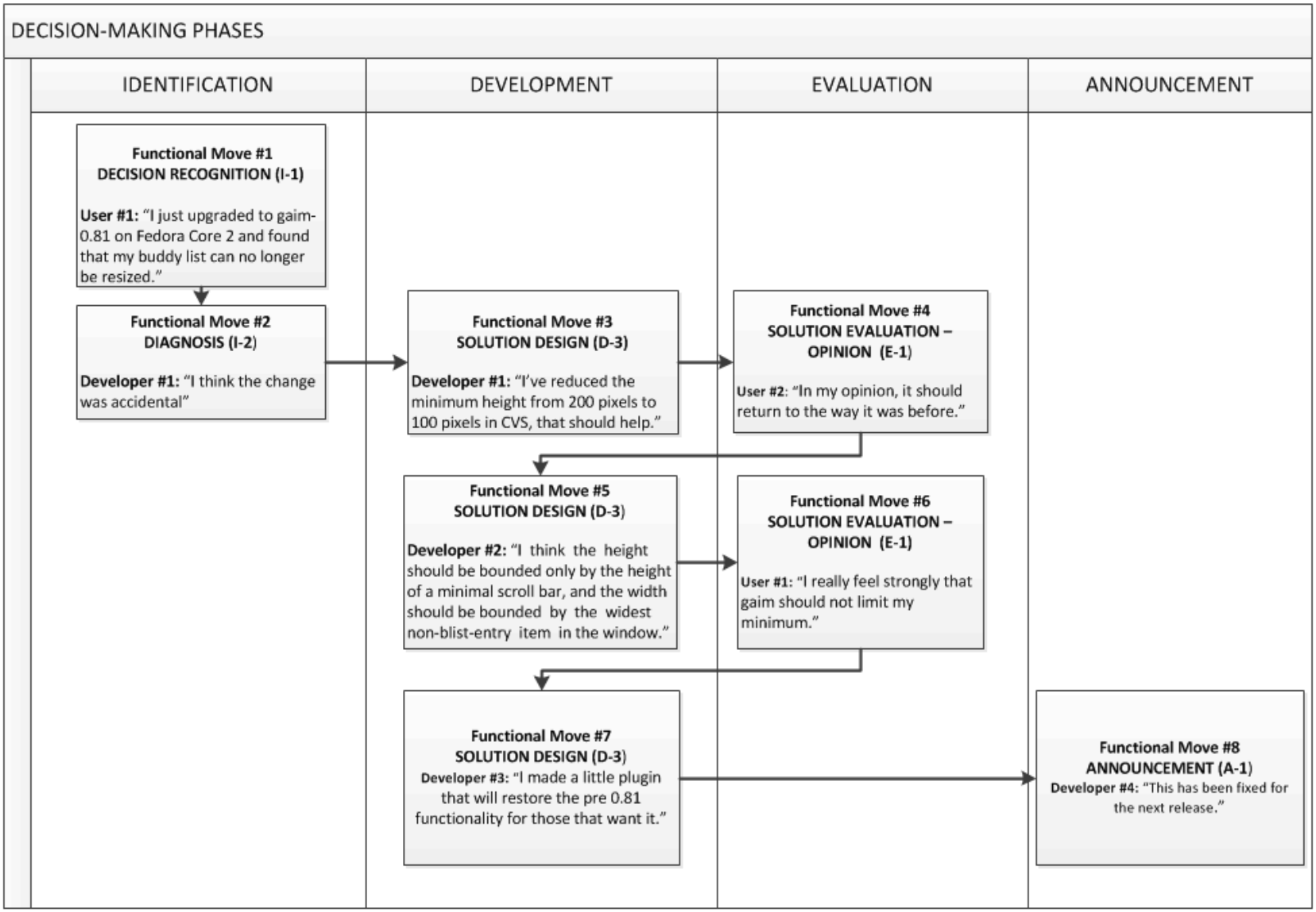

Figure 1. An Example Illustrating How a Decision Episode is Coded for Functional Moves

4.3 Findings: Qualitative Analysis of Decision-Making Patterns

Following the procedure described in section 4.2, we sorted the 300 decision-making episodes into 5 clusters according to the number of phases. We labelled these processes as short-cut, implicit-development (implicit-D), implicit-evaluation (implicit-E), complete, and abandoned decision processes (i.e., lacking a final decision announcement). Figure 2 depicts the patterns of the five processes. The dashed lines in the figures indicate points at which there might be loops, leading to iterative decision process. The loop from decision announcement to previous phases indicates that one or more intermediate decisions were announced before the decision was finalized.

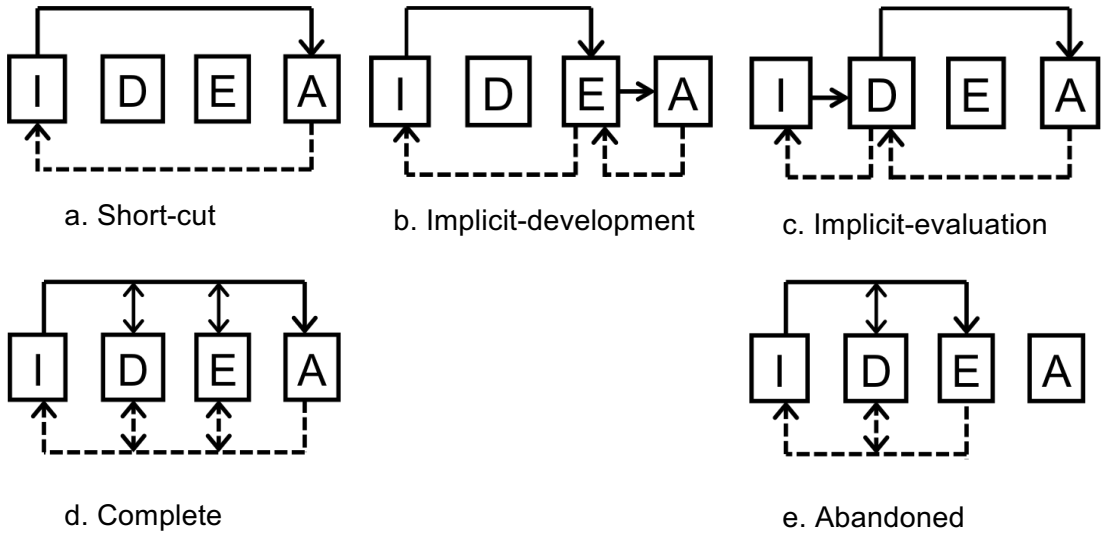

Figure 2. Five Decision-Making Processes Identified based on the Data

Short-Cut (Figure 2a). This process represents the simplest pattern, in which a decision is made right after opportunity recognition and perhaps a brief problem diagnosis, with no explicit solution development or evaluation. Examples of this kind are often observed in the bug report or problem solving discussions in software-modification decisions. For example, in one decision episode in the WebERP project, a user reported a bug (code I-1, Decision Recognition), which was quickly followed by the response of an administrator that he "*just fixed it*" (A-1, Decision Announcement), with no further discussion or evaluation. While there is an absence of team

input, we argue that these decisions are still team decisions, for two reasons: 1) Since all team members can view the bug fix and reverse it if they see it as inappropriate, a lack of reversal indicates an implicit consensus on the proposed course of action; and 2) the decision (e.g., a bug fix) affects the shared team output and binds the team to a future course of development (i.e., there are team consequences).

Implicit-D (Figure 2b). In this process, the solution development phase is skipped, which does not mean that a solution was not developed, but rather that there is a lack of evidence of the development phase in the online discussions. For example, in these episodes, the person who brings up an issue may have already done a diagnosis and provides a solution together with the issue. The subsequent discussions concentrate on evaluating the feasibility or the benefits and disadvantages of the suggested implementation, rather than looking for more alternative solutions. For example, in the aMSN project, a user wrote an email mentioning a discovered problem and providing a patch (I-1, Decision recognition): "*Unfortunately, the gnomedock was segfaulting. I am attaching a patch that fixes most (if not all) of the problems.*" An administrator mentioned that he had the same problem, and that he then applied the user's patch on his computer, which resolved the problem (E-2, Solution evaluation-action). The same administrator then said, "*I'll add patched version to CVS and thank the guy who sent the patch*" (A-1, Decision Announcement). In this example, the steps of solution analysis, search and design were not visible in the text. However, these steps were conducted at least by the user who sent the patch, and possibly by others who did not feel it was necessary to report their progress.

Implicit-E (Figure 2c). The third type of decision-making process is called "Implicit-Evaluation", indicating a lack of online evidence of evaluative discussion. In these episodes, a decision is announced directly after the solution alternatives are generated without explicit evaluation of the alternatives. For example, in aMSN, an administrator brought up a technical issue (I-1, Decision recognition) and proposed three solutions (D-3, Solution design). Most of

the subsequent messages concentrated on determining whether the problem was one for the aMSN project or just a problem from its supporting software such as a KDE problem (I-2, Diagnosis). After some discussion and testing, members confirmed it was an aMSN tray icon problem (I-2, Diagnosis). The team attention then returned to suggesting alternative solutions (D-3, Solution design) and the problem was quickly fixed (A-1, Decision announcement).

Complete (Figure 2d). In the "complete decision-making process" episodes, the team goes through all phases of decision-making, either in a linear sequence without looping back to previous phases or in an iterative sequence with loops back to previous phases, sometimes in every phase. The linear complete processes most closely resemble the rational approach described in earlier studies. For example, in the Fire project, a user reported a build failure (I-1, Decision identification). The administrator pointed out the problem immediately (I-2, Diagnosis) and provided a solution (D-3, Solution design). The user tested and confirmed the usability of the solution (E-2, Solution evaluation-action). Then the administrator promised to commit the code into CVS soon (A-1, Decision announcement).

Iterative processes were observed when the issue was more complex. The complexity of the issue stems from the fact that its diagnosis and resolution are tied to other sub-issues. As the sub-issues are interrelated, discussions may loop back to any previous phase at any time. It might sometimes be possible to find another trigger that could be interpreted as starting a new decision episode within these issues. However, since the issues are interrelated, it would not be faithful to the original source of the issue to treat them as different episodes. For example, in OFBIZ project, one administrator started a thread about how to design a workflow and based on which specifications. His first question was "*The first was, which activity should we start with, and how do we know when we're done?*" (I-1, Decision recognition). He then went on to show that he looked for existing solutions: "*I did find examples of workflows at WfMC including mail room, order processing, and various other things. It appears that the first activity for a given process is the first in the list*." (D-2, Solution search). He then described how the solution would

apply to this setting and evaluated this option, indicating it may be an easy-to-change temporary solution by saying "*At run time it will already be there so if another spec does it differently, or we find another way (the correct way?), it will be easier to change*." (E-1, Solution evaluation-option). Another administrator took the process back to the development stage by writing an example of how the start activities might work (D-3, Solution design) and then evaluated this option. The first administrator then said "*What you said about starting and ending makes a lot of sense. That's a good idea of specifying a default start activity, and for each activity specifying whether or not it can be a start activity*." (E-1, Solution evaluation, opinion) and announced the solution (A-1, Decision announcement). However, then a user jumped in to recommend an alternate solution, taking the team from decision announcement back to the solution development stage. When the administrator mentioned the user's solution would not work, the user improved his solution, leading to several loops of development and evaluation before a solution was agreed on.

Abandoned (Figure 2e). We called the final category "Abandoned decision-making process". In these processes, no decision was announced by the end of the observed decision episode. Abandonments may occur in any phase of discussion and happen for various reasons. A decision-making process may be abandoned during the identification phase due to a disagreement on whether there is a real problem or if there is a need to fix it. It may be abandoned during the development phase due to disagreement about the merits of different technical approaches and concerns. Abandonment in the evaluation phase can be due to multiple parties pursuing individual interests. For example, in the Gaim project, an administrator suggested adding audio functionality to the product (I-1, Decision recognition). Several core members challenged the availability of this functionality (I-2, Diagnosis). The discussions revealed two different preferred solutions—releasing a stable version with minor changes or releasing an unstable version with a major innovation (D-1, Solution analysis). Both sides extensively examined the current solutions, took relevant consequences into account and

provided feasible suggestions (D-3, Solution design, E-1, Solution evaluation-opinion). However, after 11 days of discussion, we found no final decision announcement (even searching the list for months after).

Table 1 shows the distribution of the five decision-making processes across the 300 decision episodes. From the table we can see that, only 38% of decisions episodes analyzed went through all four phases (labeled as "Complete"), while 52% of the discussions reached a decision while skipping one or two phases (Short-cut, Implicit-D or Implicit-E). No decision was reached in the remaining 10% of cases (Abandoned). In 23% of the decision episodes, the team made a decision right after the decision trigger was recognized (short-cut process). While 28% of decisions were made without the evaluation phase (Implicit-E process), only 1% of the decisions were made without a visible development phase (Implicit-D process).

We also clustered the decision-making processes based on the cyclicity. We found that 39% of decisions followed a linear decision process, while the other 61% included one or more loops back, following an iterative decision process.

Table 1. Count of Observed Decision Processes for All Episodes

	Short-cut	Implicit-D	Implicit-E	Complete	Abandoned	Total
Linear	56 (19%)	0 (0%)	38 (13%)	8 (3%)	14 (5%)	119 (39%)
Iterative	14 (5%)	4 (1%)	45 (15%)	105 (35%)	16 (5%)	181 (61%)
Total	70 (23%)	4 (1%)	83 (28%)	113 (38%)	30 (10%)	300 (100%)

5 PHASE 2: THE IMPACT OF CONTEXTUAL FACTORS ON THE COMPLEXITY OF DECISION PROCESSES

In the second phase of our study, we tested the theoretical hypotheses proposed in section 2.2 by performing χ^2 tests and Mann-Whitney U tests to look for the difference of complexity in the decision-making processes between tactical and strategic decision episodes, between problems and opportunities-triggered decisions, and between simple and complex software projects.

We used the data developed in phase 1 and summarized in Table 1 to test the proposed hypotheses. Since there were only 4 cases of implicit-D episodes, they were excluded from

further analysis. Moreover, because abandoned processes did not have decision announcements, it is not possible to assess the process length, one of the measures used to assess the complexity of the decision-making processes. Therefore, we also removed the 30 cases of abandoned decisions from further analysis, leaving 266 cases to be analyzed in phase 2.

5.1 Measures

The first phase of the study developed a measure for the decision-process type. In the remainder of this section, we describe how we operationalized the other constructs of the study, namely process complexity, the outcome variable, and two factors predicting complexity, namely tactical vs. strategic decisions and decision triggers. (The third hypothesized predictor, complexity of the software being developed, was varied by selection of the projects to study, as described above.)

5.1.1 Complexity

In this research, we adopted four measures from prior research to assess the complexity of decision processes. Campbell (1988) argues that any objective process characteristic that implies an increase in information load, information diversity or rate of information change can be considered as a contributor to process complexity. Based on this consideration, we first selected two measures, length and participation, to assess the complexity of decision processes. These two measures capture the number of messages (length) and the number of participants (participation) in a decision episode. We argue that as more messages and/or more participants are involved in an episode, more information in terms of quantity and diversity are contributed to the discussion, which improves the complexity of decision processes.

In addition, from a decision-process perspective, Poole and Roth (1989) propose that how often the group repeats earlier steps in the process and how many interruptions or breaks there are in group interaction determine the complexity of the decision process. Therefore, we

adopted two additional measures for complexity: the number of loops and the number of complete processes. Differences in the count of complete processes were tested using a χ^2 test, and all the other three measures are tested using a Mann-Whitney test (chosen because the data were not distributed normally). Table 2 shows the correlations among the continuous measures of complexity.

Table 2. Correlations Among the Measures of Decision Process Complexity

	1	2	3	Mean	Standard Deviation
1. Length	1			6.87	6.397
2. Participation	.862***	1		3.55	2.078
3. Number of loops	.782***	.693***	1	1.53	2.058

*** *Significant at p=.001*

5.1.2 Identification of tactical and strategic decisions

Above, we hypothesized the type of decision as a factor predicting decision-process complexity. For each episode, three researchers code each episode as either a tactical or strategic decision based on the trigger and decision announcement. Tactical decisions were identified as the team discussing and making decisions on one of the following questions, identified inductively from the analysis of messages in phase 1: 1) bug reports, 2) feature requests, 3) problem reports, 4) patch submissions and 5) to-do lists. Decision announcements for tactical decisions reflected either acceptance/rejection of a need for software code modification or acceptance/rejection of a submitted code modification.

Strategic decision were identified as discussing and making decisions on one of the following questions: 1) system design, 2) infrastructure/process, 3) business function, 4) release management and 5) other issues. Strategic decision announcements reflected either acceptance or rejection of a proposed long-term strategic plan for system design, infrastructure change and process improvement or resource allocation including task assignment and time schedule. The coders were able to code the decision types with 100% reliability.

5.1.3 Decision triggers

A second hypothesized factor affect decision-process complexity was the type of decision trigger, problem vs. opportunity. Again, for each decision episode, the three authors coded the trigger. When there are problems to deal with, e.g., that the software code does not run correctly for the developers or the users, when software bugs were identified, or when there were strategic issues that were challenges to deal with rather than opportunities (for example, when there seems to be a breach of licensing agreements), these issues were identified by coders as "problems". On the other hand, a clear identification of a desired functionality or change in the code that provides new or changed functionality and strategic issues that talked about future plans/issues about the projects were identified as an opportunity.

5.2 Findings: Quantitative Analysis of Contextual Factors on Decision Processes

We next discuss the tests of each of the hypotheses based on the data described above.

5.2.1 Tactical versus strategic decisions

To test H1, we compared decision processes for tactical and strategic decisions. Among the 266 decision-making episodes analyzed, 200 (75.2%) were tactical decisions and 66 (24.8%) were strategic decisions. A χ^2 test of the relation between decision types (tactical vs. strategic) and the frequency of the different decision processes (Table 3) indicates that there are significant differences (p**=0.008) in the decision processes adopted for these different decision types. Specifically, more strategic decisions were completed processes (59%) than for tactical decisions (38%).

Table 3. Distribution of Decision Processes between Tactical and Strategic Decisions

	Short-Cut	Implicit-E	Complete	Total
Tactical	54 (27%)	71 (35%)	75 (38%)	200 (100%)
Strategic	16 (24%)	12 (18%)	38 (59%)	66 (100%)
	70 (26%)	83 (31%)	113(43%)	266 (100%)

*χ^2 = 9.621, df = 2, p** = 0.008*

Table 4. Comparison between Tactical and Strategic Decisions

	Mean		Mann-Whitney U tests		
Measures	Tactical	Strategic	Mean Rank		p value
			Tactical	Strategic	
Length	6.04	9.36	128.09	149.90	Z=-2.013, p*=.044
Participation	3.24	4.47	126.07	156.02	Z=-2.850, p**=.004
Number of loops	1.43	1.85	131.88	138.44	Z=-.624, p=.533

*<0.05; **<0.01; ***<0.001

We used Mann-Whitney U tests to test the difference between the process complexity of tactical and strategic decisions in terms of length, participation and the number of loops. Table 4 shows the results. The results revealed that strategic decisions involved more messages (p*=.044) and more participants (p**=.004) than tactical decisions. However, the result did not show significant differences between tactical and strategic decisions in the number of loops. Still, because strategic decisions showed more complexity than tactical ones on 3 out of the 4 measures, we concluded that H1 was supported in general.

5.2.2 Problem-triggered vs. opportunity-triggered decisions

To test H2, we first analyzed tactical decisions. A χ^2 test (Table 5) indicated that there are significant differences between opportunity-triggered and problem-triggered decisions for tactical decisions (p***=0.001). Opportunity-triggered decision episodes had more completed processes (53%) than did problem-triggered episodes (28%). Meanwhile, problem-triggered episodes had more short-cut processes (33%) than did opportunity-triggered episodes (18%). Mann-Whitney U tests on length, participation and loops (Table 6) revealed that opportunity-triggered decisions involved more messages (p**=.008) and more participants (p***=.001), and had more loops (p*=.011) than problem-triggered decisions did. These analyses show that H2a was supported.

Table 5. Distribution of Decision Processes between Opportunity-triggered and Problem-triggered Decisions for Tactical Decisions

	Short-Cut	Implicit-E	Complete	Total
Opportunity	14 (17%)	24 (30%)	42 (53%)	80(100%)
Problem	40 (33%)	47 (39%)	33 (28%)	120 (100%)
	56 (27%)	72 (35%)	77(38%)	200 (100%)

*χ^2 = 13.593, df = 2, p*** = 0.001*

Table 6. Comparison between Opportunity-triggered and Problem-triggered Decisions for Tactical Decisions

	Mean		Mann-Whitney U tests		
Measures	Opportunity	Problem	Mean Rank		p value
			Opportunity	Problem	
Length	7.10	5.34	113.59	91.77	Z=-2.635, p**=.008
Participation	3.74	2.91	116.94	89.54	Z=-3.442, p***=.001
Number of loops	1.86	1.13	112.66	92.39	Z=-2.537, p*=.011

*<0.05; **<0.01; ***<0.001

Contrarily, for strategic episodes, the χ^2 test indicated no significant differences between opportunity-triggered and problem-triggered decisions (Table 7) (p=0.499). The Mann-Whitney tests (Table 8) indicated that opportunity-triggered decisions were more complex than problem-triggered only on one measure: length (p*=.046). So in general, H2b was supported.

Table 7. Distribution of Decision Processes between Opportunity-triggered and Problem-triggered Decisions for Strategic Decisions

	Short-Cut	Implicit-E	Complete	Total
Opportunity	5 (21%)	3 (12%)	16 (67%)	24(100%)
Problem	11 (26%)	9 (21%)	22 (53%)	42 (100%)
	14 (24%)	11 (18%)	36(58%)	66 (100%)

$\chi 2$ = 1.392, df = 2, p = 0.499

Table 8. Comparison between Opportunity-triggered and Problem-triggered Decisions for Strategic Decisions

	Mean		Mann-Whitney U tests		
Measures	Opportunity	Problem	Mean Rank		p value
			Opportunity	Problem	
Length	13.00	7.29	39.69	29.96	Z=-1.993, p*=.046
Participation	5.42	3.93	38.58	30.60	Z=-1.657, p=.097
Number of loops	2.79	1.31	38.98	30.37	Z=-1.826, p=.068

*<0.05; **<0.01; ***<0.001

5.2.3 Simple (IM) versus Complex (ERP) Software Projects

We turn now to H3. Because tactical decisions and strategic decisions adopt different decision processes, we decided that we should not analyze them together when considering the relation

of project types to decision-making processes. We first analyzed tactical decisions. We found that contrary to our hypothesis, IM projects and ERP projects displayed similar degree of complexity of decision processes for tactical decisions. A χ^2 test of the relationship between project type (IM/ERP) and decision processes (Table 9) showed no relation (χ^2 =1.027, p = 0.598 for tactical decisions). As well, from Table 10 we can see that IM software decisions did not show significant differences from ERP ones on either length, participation and number of loops. So for tactical decisions, H3 was not supported.

Table 9. Distribution of Decision-Making Processes between IM and ERP Projects for Tactical Decisions

	Short-Cut	Implicit-E	Complete	Total
IM	37 (29%)	44 (35%)	45 (36%)	126(100%)
ERP	17 (23%)	27 (37%)	30 (40%)	74 (100%)
	56 (27%)	72 (35%)	77(38%)	200 (100%)

χ^2 = 1.027, df = 2, p = 0.598

Table 10. Comparison between IM and ERP Projects for Tactical Decisions

	Mean		Mann-Whitney U tests		
Measures	IM	ERP	Mean Rank		p value
			IM	ERP	
Length	5.70	6.64	98.98	103.08	Z=-.488, p=.626
Participation	3.29	3.16	101.58	98.67	Z=-.360, p=.719
Number of loops	1.22	1.77	95.48	109.05	Z=-1.675, p=.094

*<0.05; **<0.01; ***<0.001

For strategic decisions, a χ^2 test (Table 11) showed that IM and ERP projects had similar patterns in the use of different decision processes (χ^2 = 4.909, p = 0.086). However, the Mann-Whitney tests (Table 12) showed that, contrary to our hypothesis, it was IM project decision processes that were more complex than ERP ones in all the three measures: length (p**=.006), participation (p**=.002) and number of loops (p*=.023). In conclusion, H3 was not supported.

Table 11. Distribution of Decision-Making Processes between IM and ERP Projects for Strategic Decisions

	Short-Cut	Implicit-E	Complete	Total
IM	6 (17%)	4 (12%)	24(71%)	34(100%)
ERP	10 (31%)	8 (25%)	14 (44%)	32 (100%)
	14 (24%)	72 (18%)	36(58%)	66 (100%)

$\chi^2 = 4.909$, $df = 2$, $p = 0.086$

Table 12. Comparison between IM and ERP Projects for Strategic Decisions

	Mean		Mann-Whitney U tests		
Measures	IM	ERP	Mean Rank		p value
			IM	ERP	
Length	12.91	5.59	39.76	26.84	Z=-2.752, p**=.006
Participation	5.56	3.31	40.34	26.23	Z=-3.040, p**=.002
Number of loops	2.59	1.06	38.49	28.20	Z=-2.266, p*=.023

*<0.05; **<0.01; ***<0.001

6 DISCUSSION

We developed two sets of insights from our analysis. First, we saw decision-making processes in FLOSS development as having multiple sequences that reflect the unique characteristics of FLOSS setting. Second, we observed that task types in terms of tactical vs. strategic decisions and problem-triggered vs. opportunity-triggered tactical decisions do have different impacts on the complexity of decision processes. In the following section, we discuss each empirically-generated insight, offer logical explanations of the results and refer to related literature to validate our argument.

6.1 Multiple Sequences of Decision-Making Processes in FLOSS Development

In phase 1 of our research, we identified 5 different decision-making processes varying in both numbers and sequences of decision-making phases: short-cut, implicit-development, implicit-evaluation, complete and abandoned processes. Four patterns were observed in the use of these processes: frequent short-cuts, frequent implicit-evaluation, infrequent implicit-development and many cycles looping back to previous stages in decision-making. We explain these patterns of different decision-making processes based on 1) the unique characteristics of FLOSS development and 2) the high level of dependency of FLOSS decision process on

information technologies.

First, we observed that the decision-making processes as exhibited in the discussion fora are unlike those observed in other decision-making contexts. For example, Mintzberg et al. (1976) argue that the evaluation-choice of a solution (evaluation in our case) must be included in any decision process. However, in our study, 23% of the decisions were made without any explicit discussion of solutions (i.e., 70 of 300 decisions were short-cut). The high frequency of short-cut decisions in what is often described as an open and participative setting is at first surprising. In addition to short-cut decisions, we found that 28% of decision episodes (a total of 83 out of 300) followed the "Implicit-Evaluation" process that skips the evaluation phase. In contrast, only 1% (4 out of 300) followed the "Implicit-Development" process, which includes an evaluation phase but skips the development phase.

At first, these results seem to be a paradox: open projects that make decisions in a seemingly opaque and non-participatory fashion. While we cannot completely rule out the existence of unarchived offline discussion that contains the missing phases, it appears that the lack of evaluation phase and other decision-making phases reflects an action orientation for decision making in FLOSS development teams: that it is preferable to simply try out a solution rather than performing detailed evaluation of potential alternatives in advance. This value is reflected in a description of the Internet Engineering Task Force decision process (part of the cultural heritage of FLOSS): "We reject: kings, presidents and voting. We believe in: rough consensus and running code" (Clark 1992). The result is a set of decision processes that emphasize making a decision rather than evaluating options.

Second, we found that developers often raise questions about others' actions based on their knowledge, leading back to previous phases of decision-making, resulting in a high proportion of cyclic processes (181 out of 300, 61%). While our findings are in line with observation that "IS decisions are often complex and dynamic" (Boonstra 2003, p.206), the factors that are previously used to explain this cyclicity, such as political influences, urgency and necessity

(Mintzberg et al. 1976, Eisenhardt and Zbaracki 1992), do not seem to apply in this setting. Rather, the dynamism of decision making in the FLOSS context seems to be an artifact of how FLOSS teams interact using ICT that allows for asynchronous communication and collaboration, meaning that anyone can observe and contribute to a decision in process, even joining later a discussion that has been going on for some time. This pattern may also reflect the fact that no individual organizes the discussions to follow a normative path, as would be observed in teams with managers or decision support systems to structure the decision process.

While in organizational settings, the dynamic nature of the decision making may to an extent indicate inefficiencies, in an open setting such as FLOSS, where decision-making speed is not necessarily a goal of the voluntary developers, the process allows participants the opportunity to jump in any time to contribute to work and related decisions, thereby increasing the level of complexity in FLOSS decision-making. In conclusion, we suggest that the cyclicity in these teams results from the self-organizing nature of the teams and the use of asynchronous communication media, rather than the factors that have been suggested to lead to cycles in other decision-making teams.

To sum up, consistent with multiple sequence models of decision-making, we found FLOSS development teams enact various decision-making processes. Further, their decision-making processes display certain patterns that we attribute to the unique characteristics of FLOSS development and the dependency on extensive ICT use.

6.2 Contextual Factors Impact the Complexity of Decision Processes

We next discuss the impacts of the different contextual factors studied on the decision-making process. First, as we expected, our results show that in the FLOSS development context, tactical decisions use less complex processes than strategic decisions do. Specifically, tactical decisions have fewer complete processes and involve fewer discussion messages and fewer participants. We explain these differences by differences in the kind of information technology support available to different kinds of decisions. To conceptualize the role of

information technology, we draw on Barcellini, et al. (2014), who identify two spaces of actions in FLOSS: the discussion space in which the participants can asynchronously communicate with each other (e.g., email lists, discussion fora and chats), and the implementation space in which developers simultaneously work on the production and documentation of the software code (e.g., Concurrent Version System (CVS), Subversion or Git, bug trackers, project documentation systems and so on). We argue that FLOSS tactical decisions about software in particular are supported by these two kinds of systems, those in the discussion space and those in the implementation space. By referring to the software code available in the implementation space, developers can apply what Bolici et al. (2015) describe as stigmergic coordination. These mechanisms thus reduce the need for extensive discussion in the discussion space. This finding is consistent with other research on FLOSS development that finds that much FLOSS software development work does not require much explicit coordination (Krishnamurthy 2002, Howison and Crowston 2014).

Further, the results show that in tactical decisions, those triggered by problems use simpler processes than those triggered by opportunities do. However, there is no significant difference in process complexity between these two types of decision triggers in strategic decisions. We argue that compared to opportunity triggers which illustrate future actions or plans, problem triggers are usually about existing software code. Acting as a communication medium, source code enables developers to examine the problems directly, thus reducing the communication effort with other developers, which in turn reduces the complexity of decision processes.

Lastly, we expected that projects that develop simpler software would use simpler decision processes than those that develop more complex software. However, contrary to our expectation, we did not find a statistically significant difference in the level of complexity of decision-making processes between simple (IM) and complex (ERP) software projects. Instead, our findings suggest that FLOSS projects tend to adopt similar decision-making processes for tactical decisions regardless of the nature of the software developed by the projects. This

similarity reflects the observation that the software development process seems to be organized similarly across projects: using same sets of ICT tools in discussion and implementation spaces, parallel development and debugging which involve loosely-centralized and gratis contribution from individual voluntary developers (Feller and Fitzgerald 2000), resulting in developers selecting similar scope of problems to work on, with similar decision demands.

Interestingly opposite our hypothesis, our research found that strategic decisions in simple (IM) projects actually exhibited more complex processes than those in complex (ERP) projects in terms of length, participation and number of loops. Our initial hypothesis was based on a consideration of the demands of the decision facing the project for increased knowledge as an input. The contrary finding leads us to speculate that the nature of the decision process might be driven instead by the supply of developers interested in the decision, which is related to the popularity of the software developed. IM clients have individuals as the users, while ERP systems have companies as the users, which might make IM projects more attractive to individual developers, necessitating an increased level of participation in IM projects. For tactical decisions, tools in the implementation space and software code itself reduce the coordination effort needed, as we argued above. On the other hand, strategic decisions receive no support from these tools, so developers have to coordinate more closely and explicitly to spell out decisions in the discussion space, which results in the observed higher complexity, as manifested by more messages posted, more participants and more loops.

7 IMPLICATIONS AND CONCLUSIONS

The primary goal of this study was to understand the decision-making process in community-based FLOSS development teams. Though the study has some limitations, it makes both theoretical and practical contributions as discussed below.

7.1 Limitations

A limitation of this study is the exclusion of synchronous discussion fora, such as Internet

Relay Chat, Instant Messaging or phone calls. It is possible that some of the steps in the decision-process that we infrequently observed were in fact carried out by a subgroup using such alternative channels. Future research should consider how decisions are made in these synchronous communication channels. However, we would argue that the use of such channels would not change our main conclusion, namely that many decisions that bind the teams to a course of action are made without explicit involvement of the entire team in seemingly important phases of the decision process.

Another limitation of this research is the small sample size (i.e., five projects and 300 decision episodes). While it enabled us to conduct manual coding and provided us with rich data that increased our understanding of the decision-making process from different projects, it limited the types of statistical analysis we could run with our data. For example, we only used two types of FLOSS projects (i.e., IM projects vs. ERP projects) to test H3, which limits our analysis of the differences of complexity between projects that develop simpler products and those develop more complex ones (or alternately, between more and less popular projects), thus limiting the generalizability of the result. Nevertheless, the decision processes and relationships we have identified provide the foundation for deeper exploration and potentially richer explanations of decision-making processes in FLOSS teams. Future research should apply the framework of this research to a larger and more representative sample of FLOSS projects.

7.2 Theoretical Contributions

Our first contribution to decision-making literature is the identification of the five different decision-making processes observed in FLOSS development teams. Identification of these processes is important because the decision process used by the group directly affects group performance. Further, our study reveals that FLOSS decision-making is not a homogeneous phenomenon. Rather, it appears to be one that invokes different processes with different level of complexity based on the nature of decisions at hand and the interests of the participants. Such

an in-depth examination of the microstructures of decision-making processes compliments existing macro-level research on decision-making (e.g. Raymond 1998, e.g. German 2003). The frequency and type of decision-making processes used by FLOSS teams can be inputs for future theory development efforts predicting group performance.

Second, our study identifies which decision types exhibit more or less complex decision-making processes. As explained further below, the complexity level of decision-making processes may influence the motivation to participate, as well as influencing the team's decision-making effectiveness and efficiency. Therefore, identification of the tools that bring about the use of relatively more complex decision-making process is an important input in the development of a more complete theory of FLOSS development, including FLOSS decision-making effectiveness and developer participation. Past research shows that task complexity determines human performance due to the demands it places on the knowledge, skills and resources of the task performers (Wood 1986). In this study, we show that the different types of tasks and the use of various information technologies change the way in which individuals manage their decision making, and thus the decision-making process complexity. Therefore, we suggest decision-making process complexity as a mediator for FLOSS research that thus far has not included this important variable.

Lastly, our results provide empirical support to the argument of stigmergic coordination in FLOSS development. By examining those decision episodes using simpler decision processes, we found that many of them had mentioned or referred to software codes explicitly in their discussion. Although prior research had proposed that stigmergic coordination makes explicit discussion unnecessary (Robles et al. 2005, Crowston et al. 2011), no empirical research has been conducted examining this question. With shared work products and discussion based on asynchronous communication, developers can work independently to determine and test solutions rather than needing to immediately discuss them with others, a decoupling that enables distributed voluntary contributors to be effective participants.

7.3 Practical Implications

Three groups of individuals in the practitioner community can benefit from the results of this study: 1) participants and leaders of FLOSS teams; 2) managers and members of companies who would like to actively contribute to existing FLOSS teams or to develop and support such teams; and 3) those who would like to apply the FLOSS model of work in their organizations.

First, our results suggest that the complexity of tactical decision-making processes is quite different from that of strategic decisions. Therefore, FLOSS development teams might want to consciously adopt different strategies when making different types of decisions. For tactical decisions, the development tools adopted by FLOSS teams greatly reduced the need for explicit coordination and communication among multiple developers, allowing members to perform tasks in an independent fashion. Contrariwise, for strategic decisions, FLOSS development teams might want to fully employ asynchronous discussion to encourage participation and reflection. Understanding decision-making processes also enables the creation of group decision support systems and other information systems that would fulfill the team requirements. Discussion and implementation spaces are especially crucial to the success and continuity of distributed teams such as FLOSS, which depend on such systems for both task accomplishment and group maintenance.

Second, the success of FLOSS development has attracted more and more companies' active participation (Dahlander and Magnusson 2008). Companies first need to understand how FLOSS communities operate before they can be successfully involved in FLOSS development. By understanding the decision-making processes in FLOSS teams, firms can know better what kind of decision processes would likely be used for different task types, so they can adjust their behaviors to better contribute to FLOSS development.

Third, though this research studied decision-making processes in FLOSS development teams, many of our findings can be applied to self-organizing organizational virtual teams, and similar open organizations more generally. Indeed, Markus et al. (2000) argue that,

Although managers in industries other than software development may prefer more traditional styles of management, they should remember that the world is changing and workers are changing along with it. In a labor force of volunteers and virtual teams, the motivational and self-governing patterns of the open source movement may well become essential to business success (p. 25).

The results of this study offer several practical insights that can benefit organizations in decision making in a distributed, self-organizing, open work environment. For example, managers should consider implementing tools that enable team members to coordinate through their work product, and augment these with discussion tools in a way that mirrors the FLOSS practice. For example, co-workers may be able to substitute examination of shared documents (e.g., with tools such as Google Documents or Lotus Notes) for extensive discussion of their contents in the discussion space, and rely on self-organized contribution to the shared work rather than detailed negotiation about who will take on which task. In this way the apparent advantages of FLOSS development may become more widely available.

REFERENCES

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

Amrit, C. and J. Van Hillegersberg (2010). "Exploring the impact of socio-technical core-periphery structures in open source software development." journal of information technology **25**(2): 216-229.

Annabi, H., K. Crowston and R. Heckman (2008). Depicting what really matters: Using episodes to study latent phenomenon. International Conference on Information Systems (ICIS), Paris, France Year.

Barcellini, F., F. Détienne and J.-M. Burkhardt (2014). "A situated approach of roles and participation in Open Source Software Communities." Human–Computer Interaction **29**(3): 205-255.

Boldyreff, C., K. Crowston, B. Lundell and T. Wasserman (2009). "Measuring potential user interest and active user base in FLOSS projects." Skovde, Sweden Sprouts: Working Papers on Information Systems **9**(33): http://sprouts.aisnet.org/9-33.

Bolici, F., J. Howison and K. Crowston (2015). "Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms." Cognitive Systems Research.

Boonstra, A. (2003). "Structure and analysis of IS decision-making processes." European Journal of Information Systems **12**(3): 195–209.

Campbell, D. J. (1988). "Task Complexity: A Review and Analysis." Academy of Management Review **13**(1): 40-52.

Clark, D. D. (1992). A Cloudy Crystal Ball: Visions of the Future. Twenty-Fourth Internet Engineering Task Force. Cambridge, MA**:** 540–543.

Clarke, P. and R. V. O'Connor (2012). "The situational factors that affect the software development process: Towards a comprehensive reference framework." Information and Software Technology **54**(5): 433-447.

Colombo, M. G., E. Piva and C. Rossi-Lamastra (2014). "Open innovation and within-industry diversification in small and medium enterprises: The case of open source software firms." Research Policy **43**(5): 891–902.

Crowston, K. (2000). Processes as theory in information systems research. IFIP TC8 WG8.2 International Working Conference on the Social and Organizational Perspective on Research and Practice in Information Technology, Arlborg, Denmark, Kluwer Academic Year.

Crowston, K., C. Østerlund, J. Howison and F. Bolici (2011). Work as coordination and coordination as work: A process perspective on FLOSS development projects. Third International Symposium on Process Organization Studies. Corfu, Greece.

Crowston, K., K. Wei, J. Howison and A. Wiggins (2012). "Free/Libre open-source software development: What we know and what we do not know." ACM Computing Surveys (CSUR) **44**(2): 7.

Dahlander, L. and M. Magnusson (2008). "How do Firms Make Use of Open Source Communities?" Long Range Planning **41**(6): 629-649.

Drury, M., K. Conboy and K. Power (2012). "Obstacles to decision making in Agile software development teams." Journal of Systems and Software **85**(6): 1239-1254.

Economides, N. and E. Katsamakas (2006). "Two-Sided Competition of Proprietary vs. Open Source Technology Platforms and the Implications for the Software Industry." Management Science **52**(7): 1057–1071.

Eisenhardt, K. M. and M. J. Zbaracki (1992). "Strategic decision making." Strategic Management Journal **13**: 17–37.

Eseryel, U. Y. (2014). "IT-enabled knowledge creation for open innovation." Journal of the Association for Information Systems **15**(11): 355-432.

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

Feller, J. and B. Fitzgerald (2000). A framework analysis of the open source software development paradigm. twenty first international conference on Information systems, Brisbane, Australia, Association for Information Systems Year.

Fitzgerald, B. (2006). "The transformation of Open Source Software." MIS Quarterly **30**(4).

Gacek, C. and B. Arief (2004). "The many meanings of Open Source." IEEE Software **21**(1): 34–40.

German, D. M. (2003). "The GNOME project: A case study of open source, global software development." Software Process: Improvement and Practice **8**(4): 201–215.

Guzzo, R. A. and E. Salas (1995). Team Effectiveness and Decision Making in Organizations. San Francisco, CA, Jossey-Bass 1995.

Hackman, R. (1990). Groups that work (and those that don't): Creating conditions for effective teamwork. San Francisco, Jossey-Bass 1990.

Haefliger, S., G. von Krogh and S. Spaeth (2008). "Code Reuse in Open Source Software." Management Science **54**(1): 180-193.

Herring, S. C., Ed. (1996). Computer-Mediated Communication: Linguistic, Social, and Cross-Cultural Perspectives. Philadelphia, John Benjamins 1996.

Howison, J. and K. Crowston (2014). "Collaboration through open superposition: A theory of the open source way." Mis Quarterly **38**(1): 29-50.

Huber, G. P. (1990). "A Theory of the Effects of Advanced Information Technologies on Organizational Design, Intelligence, and Decision Making." The Academy of Management Review **15**(1): 47-71.

Keen, P. G. W. and J. J. Cummins (1994). Networks in Action. Belmont, California, Wadsworth 1994.

Kerr, N. L. and R. S. Tindale (2004). "Group performance and decision making." Annual Review of Psychology **55**: 623–655.

Kiesler, S. and L. Sproull (1992). "Group Decision Making and Communication Technology." Organizational Behavior and Human Decision Processes **52**: 96-123.

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

Markus, M. L., B. Manville and E. C. Agres (2000). "What makes a virtual organization work?" Sloan Management Review **42**(1): 13-26.

Mennecke, B. E., J. S. Valacich and B. C. Wheeler (2000). "The Effects of Media and Task on User Performance: A Test of the Task-Media Fit Hypothesis." Group Decision and Negotiation **9**: 507-529.

Miller, K. (2008). Organizational Communication: Approaches and Processes. Bonston, MA, Wadsworth Cengage Learning 2008.

Mintzberg, H. (1973). The Nature of Managerial Work. New York, Harper & Row 1973.

Mintzberg, H., D. Raisinghani and A. Theoret (1976). "The Structure of "Unstructured" Decision Process." Adminstrative Science Quarterly **21**(2): 246-275.

Moe, N. B., A. Aurum and T. Dybå (2012). "Challenges of shared decision-making: A multiple case study of agile software development." Information and Software Technology **54**(8): 853-865.

Moon, J. Y. and L. Sproull (2000). "Essence of distributed work: the case of Linux kernel." First Monday **5**(11).

Morner, M. and G. von Krogh (2009). "A note on knowledge creation in open-source software projects: What can we learn from Luhmann's theory of social systems?" Systemic Practice and Action Research **22**(6): 431-443.

Nelson, M., R. Sen and C. Subramaniam (2006). "Understanding open source software: A research classification framework." Communications of the Association for Information Systems **17**(1): 12.

Niederman, F., A. Davis, M. E. Greiner, D. Wynn and P. T. York (2006). "A research agenda for studying open source I: A multi-level framework." Communications of AIS **2006**(18): Article 7.

Nutt, P. C. (1984). "Types of organizational decision processes." Administrative Science Quarterly **29**(3): 414-450.

Oh, W. and S. Jeon (2007). "Membership Herding and Network Stability in the Open Source Community: The Ising Perspective " management Science **53**(7): 1086-1101.

Park, J.-G. and J. Lee (2014). "Knowledge sharing in information systems development projects: Explicating the role of dependence and trust." International Journal of Project Management **32**(1): 153-165.

Payne, J. W. (1976). "Task Complexity and Contingent Processing in Decision making: An Information Search and Protocol Analysis." Organizational Behavior and Human Performance **16**(2): 366-387.

Poole, M. S. (1983). "Decision development in small groups II: A study of multiple sequences in decision making." Communication Monographs **50**(3): 206-232.

Poole, M. S. and C. L. Baldwin (1996). Developmental processes in group decision making. Communication and Group Decision Making. R. Y. Hirokawa and M. S. Poole. Thousands Oaks, CA, SAGE 1996**:** 215-241.

Poole, M. S. and M. E. Holmes (1995). "Decision development in computer-assisted group decision-making." Human Communication Research **22**(1): 90–127.

Poole, M. S. and J. Roth (1989). "Decision Development in Small Group IV: A typology of Group Decision Paths." Human Communication Research **15**(3): 323-356.

Poole, M. S., D. R. Seibold and R. D. McPhee (1985). "Group decision-making as a structurational process." Quarterly Journal of Speech **71**(1): 74–102.

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

Raymond, E. S. (2001). The Cathedral and the Bazaar: Musings of Linux and Open Source from an Accidental Revolutionary. Sebastapol, CA, O'Reilly and Associates 2001.

Robey, D., H. M. Khoo and C. Powers (2000). "Situated-learning in cross-functional virtual teams." IEEE Transactions on Professional Communication(Feb/Mar): 51–66.

Robles, G. (2004). "A software engineering approach to libre software." Open Source Yearbook.

Robles, G., J. J. Merelo and J. M. Gonzalez-Barahona (2005). "Self-organized development in libre software: a model based on the stigmergy concept." ProSim'05: 16.

Ryan, H. W. (1999). "Managing development in the era of large complex systems." Information Systems Management **16**(89-91).

Santos, C., G. Kuk, F. Kon and J. Pearson (2013). "The attraction of contributors in free and open source software projects." The Journal of Strategic Information Systems **22**(1): 26-45.

Scacchi, W. (2007). "Free/Open Source Software Development: Recent Research Results and Methods." Advances in Computers **69**: 243–295.

Shaikh, A. A. and H. Karjaluoto (2015). "Making the most of information technology & systems usage: A literature review, framework and future research agenda." Computers in Human Behavior **49**: 541-566.

Smart, C. and I. Vertinsky (1977). "Designs for cirisis decision units." Administrative Science Quarterly **22**(1): 640-657.

Speier, C., I. Vessey and J. S. Valacich (2003). "The Effects of Interruptions, Task Complexity, and Information Presentation on Computer-Supported Decision-Making Performance." Decision Sciences **34**(4): 771-796.

Straus, S. G. and J. E. McGrath (1994). "Does the Medium Matter? The Interaction of Task Types and Technology on Group Performance and Member Reactions." Journal of Applied Psychology **79**(1): 87-97.

von Hippel, E. and G. von Krogh (2003). "Open source software and the "private-collective" innovation model: Issues for organization science." Organization Science **14**(2): 209-223.

von Krogh, G. and E. A. von Hippel (2006). "The promise of research on open source software." Management Science **52**(7): 975–983.

Watson-Manheim, M. B., K. M. Chudoba and K. Crowston (2002). "Discontinuities and continuities: A new way to understand virtual work." Information, Technology and People **15**(3): 191–209.

Wittenbaum, G. M., A. B. Hollingshead, P. B. Paulus, R. Y. Hirokawa, D. G. Ancona, R. S. Peterson, K. A. Jehn and K. Yoon (2004). "The functional perspective as a lens for understanding groups." Small Group Research **35**(1): 17–43.

Wood, R. E. (1986). "Task complexity: Definition of the construct." Organizational Behavior and Human Decision Processes **37**(1): 60-82.

Appendix 1. The Process of Revising the Coding Scheme from Literature

First, we removed the moves “screening issues” and “authorizing decisions”, which occur frequently in traditional decision-making contexts but that we found rarely in our context. The first code seemed to be rare because in the FLOSS context, with distributed leadership, there was not a specific person in charge of decision-making process who might screen issues as needing or not needing discussion. Instead, discussions usually started immediately after an alternative was proposed. Similarly, a decision generally did not need to be authorized by a certain person or institution. In the very few cases when it did, for example, where a discussed issue needed to be handled by the administrator or the project leader, the authorization move might have been activated, but due to low occurrences, we decided not to include it in our coding scheme.

Second, we divided the move “Solution evaluation” into two functional moves: “Solution evaluation-opinion” and “Solution evaluation-action”. Solution evaluation-opinion refers to giving an opinion on the proposed option. Solution evaluation-action is an evaluation behavior that is uniquely different in asynchronous collaboration, where the team members test a proposed solution and post the results of their actions rather than simply posting opinions (Keen and Cummins 1994). In a synchronous discussion, participants rarely have time to take such action during a meeting.

The final coding scheme for stages in the decision-making process is presented below.

Phase	Functional Move	Explanation	Examples from Literature
(I) Identification	(I-1) Decision recognition routine	This move recognizes an opportunity that may lead to a decision. Triggers for software-related decisions may include whether a fix is needed. Secondly a patch that is sent to the team may initiate an opportunity for decisions.	problem analysis (Poole and Roth 1989); decision recognition (Mintzberg et al. 1976)
	(I-2) Diagnosis	This move focuses on understanding the underlying reasons that cause problems or create opportunities for decisions. It also includes asking and providing background information, such as installation environment, computer configuration etc.	problem critique (Poole and Roth 1989); diagnosis (Mintzberg et al. 1976)
(D) Development	(D-1) Solution analysis	This move describes the activities trying to develop its solution in general terms, rather than providing specific solutions, such as team rule/norm, criteria and general directions to guide the solution.	solution analysis (Poole and Roth 1989)
	(D-2) Solution search	This move describes the activities trying to look for ready-made solutions based on experiences and existing resources, rather than designing solution by themselves.	search (Mintzberg et al. 1976), solution search (Poole and Roth 1989)
	(D-3) Solution design	This move describes the activities designing and providing specific solutions and suggestions by themselves, or modifying the ready-made/existing ones according to the new context.	design (Mintzberg et al. 1976), solution elaboration (Poole and Roth 1989)
(E) Evaluation	(E-1) Solution evaluation-opinion	This move explicit or implicitly comments on potential alternatives, based on personal experiences/ preferences, rather than real testing/checking.	evaluation-choice (Mintzberg et al. 1976); solution evaluation (Poole and Roth 1989)
	(E-2) Solution evaluation-action	This move explicit or implicitly comments on potential alternatives, based on actual testing/checking. It also includes describing the details how the alternatives are tested and what results come out of that.	[Emergent code grounded in the data, non-existent in the literature]
	(E-3) Solution confirmation	This move describes the activity to ask for confirmation or initiate voting.	solution confirmation (Poole and Roth 1989)
(A) Announcement	(A-1) Decision announcement	This move announces the final decision on team level.	decision product (Wood 1986)