Communications of the Association for Information Systems

Volume 46 Article 20

4-2020

Decision-making Processes in Community-based Free/Libre Open Source Software-development Teams with Internal Governance: An Extension to Decision-making Theory

U. Yeliz Eseryel
East Carolina University, yeliz@eseryel.com

Kangning Wei
Shandong University

Kevin Crowston
Syracuse University

Recommended Citation
Eseryel, U., Wei, K., & Crowston, K. (2020). Decision-making Processes in Community-based Free/Libre Open Source Software-development Teams with Internal Governance: An Extension to Decision-making Theory. Communications of the Association for Information Systems, 46, pp-pp. https://doi.org/10.17705/1CAIS.04620

Communications of the Association for Information Systems

Research Paper DOI: 10.17705/1CAIS.04620 ISSN: 1529-3181

Decision-making Processes in Community-based Free/Libre Open Source Software-development Teams with Internal Governance: An Extension to Decision-making Theory

U. Yeliz Eseryel
Management Information Systems
East Carolina University
yeliz@eseryel.com

Kangning Wei
School of Management
Shandong University

Kevin Crowston
School of Information Studies
Syracuse University

Abstract:

Community-based free/libre open source software (FLOSS) teams with internal governance constitute an extreme example of distributed teams, prominent in software development. At the core of distributed team success lies team decision making and execution. However, in FLOSS teams, one might expect the lack of formal organizational structures to guide practices and reliance on asynchronous communication to make decision making problematic. Despite these challenges, many effective FLOSS teams exist. We lack research on how organizations make IS development decisions in general and on FLOSS decision-making models in particular. The decision-making literature on FLOSS teams has focused on the distribution of decision-making power. Therefore, it remains unclear which decision-making theories fit the FLOSS context best or whether we require novel decision-making models. We adopted a process-based perspective to analyze decision making in five community-based FLOSS teams. We identified five different decision-making processes, which indicates that FLOSS teams use multiple processes when making decisions. Decision-making behaviors remained stable across projects even though they required different types of knowledge. We help fill the literature gap about which FLOSS decision mechanisms one can explain using classical decision-making theories. Practically, community and company leaders can use knowledge of these decision processes to develop infrastructure that fits FLOSS decision-making processes.

Keywords: Decision Making, Decision Process, Free/Libre Open Source Software Development, OSS, F/LOSS.

This manuscript underwent peer review. It was received 02/13/2019 and was with the authors for 5 months for 2 revisions. Matt Germonprez served as Associate Editor.

1 Introduction

In this paper, we identify decision-making processes in community-based free/libre open source software (FLOSS) development teams with internal governance. Given decision making constitutes an essential component of team behavior (Guzzo & Salas, 1995), researchers have extensively studied it. We need to understand decision-making processes in teams because their effectiveness can have a large impact on overall team performance (Hackman, 1990). Decision making has particular relevance to information systems (IS) research because advanced information and communication technologies (ICT) often support and influence these processes (Huber, 1990; Shaikh & Karjaluoto, 2015).

Researchers have examined FLOSS as an important phenomenon in its own right and as a potential influence on the larger IS information systems domain (Niederman, Davis, Greiner, Wynn, & York, 2006b). Researchers have said FLOSS to enable small businesses and users to play a role in democratized business software innovation in business ecosystems (Allen, 2012). Andriole (2012) suggests that FLOSS impacts the software development process due to the open architectures it encourages, which means that it creeps into every layer of the software stack. FLOSS provides many benefits and challenges to software developers and users compared to off-the-shelf software. Its benefits include higher reliability, improved security, and low cost, whereas its challenges include few or no deadlines for implementing feature or bug fixes, and a high entry barrier for non-technical users (Almarzouq, 2005).

Nelson, Sen, and Subramaniam (2006) identified that “a significant opportunity exists for studying the evolution of coordination mechanisms in FLOSS projects” (p. 278). Given that FLOSS teams follow dynamic and consensus-driven decision-making structures (Nelson et al., 2006), we follow Nelson et al.’s (2006) recommendation to investigate the extent to which one can explain FLOSS decision-making mechanisms using classical theories from organizational structure or whether they require new thinking.

However, FLOSS teams differ from one another; that is, they come in many different types, which may affect their governance and decision-making structures. As such, we need to bind our study. West and O'Mahony (2008) identified two types of FLOSS communities: autonomous and self-managed communities versus sponsored communities. Di Tullio and Staples (2013) proposed a finer division into three types/phases of FLOSS governance as de Laat (2007) identified: 1) spontaneous governance, 2) internal governance, and 3) governance towards outside parties. Spontaneous governance refers to small, self-directing projects that have no explicit or formal control or coordination mechanisms. Internal governance refers to projects that have existed for a period of time, that have multiple participants, and that require coordination and control to achieve desired outcomes (Midha & Bhattacherjee, 2012). However, as their name suggests, these projects feature internal governance. Governance towards outside parties refers to highly institutionalized projects, which occur when non-profit foundations protect projects or when projects work with companies, which means that those parties determine governance. In our study, we focus on the second type (i.e., internal governance) because they have reached a big enough size such that they feature identifiable (yet possibly still emergent) decision processes. We refer to such FLOSS teams as “community-based FLOSS teams with internal governance”. Henceforth in this paper, we use the terms FLOSS teams and community-based FLOSS teams with internal governance interchangeably for brevity.

Our interest in understanding the decision-making process in community-based FLOSS teams with internal governance arose due to three distinct characteristics of the FLOSS context that we expected would pose barriers to effective decision making and, thus, require novel decision-making processes.

First, community-based FLOSS teams with internal governance generally have a virtual nature since developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily via ICT (Crowston, Wei, Howison, & Wiggins, 2012). The extensive use of ICT changes the way members can interact and, thus, how they make decisions (Kiesler & Sproull, 1992). A lack of shared context and numerous discontinuities in communication that virtual team members face can hamper decision making (Watson-Manheim, Chudoba, & Crowston, 2002). While FLOSS teams with governance toward outside parties, such as projects under the Apache Foundation or company-based FLOSS projects such as Red-Hat, have a shared context and shared norms, community-based FLOSS teams lack these commonalities that help ease common work.

Community-based FLOSS teams with internal governance feature not only technologically mediated communication but also technologically mediated work. Further, prior research has suggested that information technologies can affect the decision-making mechanisms (Kiesler & Sproull, 1992).

Asynchronous communications make it impossible for participants to catch cues available in synchronous media such as voice tone, speed, and body language. The lack of such cues may create barriers to decision-making process since sensemaking and understanding become more difficult for participants. On the other hand, we do not know to which extent asynchronous decision making may allow participants to use novel decision-making processes.

Second, given work's distributed nature in FLOSS teams, decision-making processes must enable participants from all around the world and, thus, from different time zones to contribute. For instance, decision-making processes in FLOSS teams usually have an asynchronous nature. Further, such teams particularly rely on information technologies, such as team discussion forums, websites, bug trackers and source code repositories—what Barcellini, Détienne, and Burkhardt (2014) term discussion spaces—to communicate, coordinate, and discuss alternatives. The asynchronous collaboration provides an additional unexpected benefit. In FLOSS teams, interested individuals can widely access the knowledge base for the decision and the decision-making actions and contribute with their opinions and knowledge with minimal barriers. In comparison to traditional organizations, researchers have found that more people in FLOSS development can share power and participate in the team's activities (Crowston et al., 2012). The more participants and more discussions in decision-making processes, the more knowledge that others can see and access, which, in turn, enables participants to work on their own and contribute what they have done back to the team.

Third, unlike organizational teams, researchers have identified community-based FLOSS teams with internal governance as self-organizing, autonomous, and self-managed (West & O'Mahony, 2008), which means that they lack formal management authority relations (O'Mahony & Ferraro, 2004a, 2007) (i.e., they do not have externally appointed leaders). Indications of ranks or roles materialize through interaction rather than external cues, which means that FLOSS teams have no hierarchical source of decision authority. In these teams, leadership emerges naturally and fluidly in that individuals gain or lose leadership through their actions over time (Eseryel & Eseryel, 2013). FLOSS members, similar to most members in engineering settings, value technical contributions over all else and often eschew positional power. Eseryel and Eseryel (2013) found that leaders in FLOSS teams provide action-embedded transformational leadership, which means that they "emerge as leaders through their consistently noteworthy contributions to their team over extended periods of time and through the inspiration they provide other team members" (p. 108). Accordingly, decision making in this setting has even more importance since decisions will likely contribute to leadership emergence and provide the basis for organizing. Technical contributions that other team members value (such as the number and popularity of the packages one maintains) can determine one's membership and leadership decisions. Yet, when communities impose rules on existing or interested members such as face-to-face meetings, key signings, and recommendations by existing members for membership, such as in the case of the Debian community, they may influence the relative influence that technical contributions have on membership and leadership and introduce new factors such as the centrality in face-to-face networks as important determinants (O'Mahony & Ferraro, 2004a, 2007).

Niederman et al. (2006b) suggest a multi-level approach to investigating FLOSS; namely, at the group, project, and community levels. Further, they recommend that researchers study the mechanics for creating artifacts. In this paper, following their advice, we investigate the decision-making mechanics for software development. Examining how the FLOSS community adapts decision-making processes 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 has become increasingly virtual. At a more specific level, knowledge of the FLOSS decision-making process can be informative for organizations or firms collaborating with FLOSS teams (Santos, Kuk, Kon, & Pearson, 2013). FLOSS impacted the software industry significantly, and many organizations develop and/or use FLOSS (Ven & Verelst, 2011). As organizations do not passively engage in FLOSS development (Colombo, Piva, & Rossi-Lamastra, 2014), organizations that hope to extract the most value from their interactions with FLOSS communities need to understand their decision-making processes. Our investigation into FLOSS decision making concurs with Feller and Fitzgerald's (2000) recommended research agenda to focus on FLOSS communities' development processes based on the traditional reporting questions such as who, what, where, when, why and how.

While prior literature has recognized decision making as an important function in FLOSS teams (Crowston et al., 2012), it has black-boxed FLOSS-related decisions and primarily provided the decisions' outcomes

rather than investigating the decision-making process. To fill this gap, we explore decision-making processes' structure in community-based FLOSS development teams. More specifically, based on the contingency model of decision-making processes (which we discuss in detail in Section 2), we answer the following research question (RQ):

RQ: What decision-making processes emerge in community-based FLOSS development teams with internal governance?

To answer this research question, we analyze decision episodes from five FLOSS teams to identify distinct decision-making patterns.

2 Theoretical Background

In this section, we first position our research in the extant FLOSS research. Then, we introduce phase theories of team decision-making processes, which we use to analyze FLOSS data.

2.1 Extant FLOSS Research and Decision Making

To position our research in the current FLOSS literature, we briefly review the FLOSS literature, and we discuss how each stream has addressed decision making. Much research has addressed open source software, and numerous review papers summarize the state of FLOSS research at various points in time (e.g., Aksulu & Wade, 2010; Crowston et al., 2012; Nelson et al., 2006; Niederman, Davis, Greiner, Wynn, & York, 2006a; Scacchi, 2007; von Krogh & von Hippel, 2006). We categorize FLOSS research into six areas: 1) FLOSS as an example of a unique phenomenon; 2) country-, industry-, and market-level investigations; 3) company-level decisions; 4) project-level processes and decisions; 5) inter-project influences; 6) individual-level decisions. Our research falls into the fourth category.

We define the first stream of FLOSS research as "FLOSS as an example of a unique phenomenon". Some examples include Love and Hirschheim's (2017) conceptualizing FLOSS as exemplifying the emerging genre of crowdsourced research genre. Similarly, Pykäläinen, Yang, and Fang (2009) defined FLOSS strategy as a novel strategy and identified conditions where this strategy would be viable. von Krogh (2009) used FLOSS phenomenon to illustrate how, in developing theory about knowledge, one needs both individualist and collectivist perspectives on the locus of knowledge. Barrett, Heracleous, and Walsham (2013) approached FLOSS diffusion as an IT-related innovation, a computerization movement.

We classify the second stream of FLOSS research as macro-level FLOSS research, which means country-, industry-, and market-level research on FLOSS. For example, Maldonado (2010) identified the process of FLOSS adoption and innovation at the country level with a case study on Venezuela. Deodhar, Saxena, Gupta, and Ruohonen (2012) identified the emergent hybrid business models that software product vendors use as a result of combining FLOSS models and their existing business models.

The third stream of FLOSS research focuses on company-level decisions—strategic decisions about a range of issues at the company level including strategy determination, value creation, licensing, FLOSS acquisition and adoption, and the use of employee time and skills. For example, Morgan, Feller, and Finnegan (2013) and Morgan and Finnegan (2014) theorized about OSS-based value creation and value capturing using inter-organizational networks. Alspaugh, Scacchi, and Asuncion (2010) provided guidance for achieving best-of-breed component strategy while obtaining desired license rights when one combines FLOSS software and proprietary software development efforts. Singh and Phelps (2013) identified the factors that influence FLOSS licensing decisions. Mehra, Dewan, and Freimer (2011) and Mehra and Mookerjee (2012) developed analytical models to support employment contract decisions to combine FLOSS participation and wage payments. Benlian (2011) developed a framework for identifying how IS managers make acquisition decisions for FLOSS versus traditional software or on-demand software. Macredie and Mijinyawa (2011) developed a framework on SMEs' OSS adoption decisions. Marsan, Pare, and Beaudry (2012) investigated IT specialists' perceptions and backgrounds that affect FLOSS adoption decisions. Feller, Finnegan, and Nilsson (2011) found four typologies for how managers in public organizations can adopt FLOSS innovation processes. Chengalur-Smith, Sidorova, and Daniel (2010) showed how infrastructure source openness influences FLOSS technology use decisions, which, in turn, increases business value. Machado, Raghu, Sainam, and Sinha (2017) discussed how the existence of FLOSS alternatives affects firms' pricing strategies and piracy-control efforts. Similarly, August, Shin, and Tunca (2013) developed an economic model to jointly analyze software originators' and subsequent FLOSS contributors' investment and pricing decisions.

The fourth stream of FLOSS research investigates various processes at the project level (our focus in this study). This research steam focuses on determinants of project success, project attractiveness, and the various processes that projects use such as innovation, knowledge creation, and requirements engineering. Much research at the project level has focused on FLOSS development processes (e.g., Howison & Crowston, 2014; Wang, Kuzmickaja, Stol, Abrahamsson, & Fitzgerald, 2014; Wei, Crowston, Li, & Heckman, 2014). Others, such as Daniel and Stewart's (2016) study, identified sources for project success when FLOSS projects share key resources such as developer attention and knowledge. They found that software coupling, interactive discussion, and externally focused developer attention directly impact completed code commits. In investigating project success, Daniel, Midha, Bhattacherjee, and Singh (2018) showed that participant differences (language, role, and contribution) and project differences (development environment and connectedness) have main and moderating effects on project success. Eseryel and Eseryel (2013) discussed how individuals emerge as leaders in FLOSS projects exhibit transformational FLOSS leadership and, thereby strategically influence systems development. Setia, Rajogopalan, and Sambamurthy (2012) showed that peripheral developers contribute to software product quality and diffusion. Santos et al. (2013) developed a theoretical model that identifies the contextual and causal factors that determine project attractiveness (source code contribution, software maintenance, and usage). They found that factors such as license restrictiveness and available resources directly influence the number of work activities in projects. Xiao, Lindberg, Hansen, and Lyytinen (2018) investigated the requirements-engineering process and showed how the distributed, dynamic, and heterogeneous structure that underlies FLOSS influences the mechanisms for managing requirements.

In this stream, one can find prior FLOSS decision-making studies at the project level or at the inter-project level. However, they do not "open the black box" to examine in detail the process that the developers use to make technical and strategic decisions about software development. The research on decisions instead typically examines governance, leadership, and authority. For example, studies have examined the distribution of decision-making power (e.g., Fitzgerald, 2006; German, 2003) and found that participants nearer to the core have greater control and discretionary decision-making authority compared to participants further from the core. O'Mahony and Ferraro (2004a, 2007) found that centrality in the face-to-face network and, to a lesser degree, technical contributions determine team membership—the basis to make certain type of decisions with respect to the software code.

Research has further categorized different governance mechanisms and approaches to leadership in different FLOSS teams. Gacek and Arief (2004) observed a connection between a hierarchical governance structure and centralized decision-making processes. Researchers have characterized the centralized decision-making process in the Linux kernel (Moon & Sproull, 2000) as a benevolent dictatorship (Raymond, 1998). In contrast, the relatively non-hierarchical GNOME team has a decentralized decision-making process that involves task forces (German, 2003). Finally, researchers have observed roles and decision-making structures to be dynamic (Nelson et al., 2006; Raymond, 2001; Robles, 2004) and fluid (O'Mahony & Ferraro, 2004b). Fitzgerald (2006) suggested that, early in a team's life, a small subset of the community will control decision making, but, as the software grows, more developers will get involved.

As FLOSS developers tend to work on multiple projects, the fifth stream of FLOSS research focuses on the influence that membership in multiple FLOSS projects has on project level-outcomes. For example, Singh, Tan, and Mookerjee (2011) showed how a project's internal and external cohesion influence 1) the project success and 2) the project's external network's technological diversity. Peng and Dey (2013) found that co-membership among project teams constitutes an effective mechanism for building network ties for knowledge sharing and further specified that leader-follower and follower-leader network ties benefit OSS success more than other types of ties. Chua and Yeow (2010) investigated the coordination process in cross-project FLOSS development and the role of development artifacts.

The last and the sixth stream of FLOSS research investigates FLOSS decisions at the individual level. The most common type of research in this category focuses on participants' motivation (Benbya & Belbaly, 2010; von Krogh, Haefliger, Spaeth, & Wallin, 2012) and commitment (Bateman, Gray, & Butler, 2011; Daniel, Maruping, Cataldo, & Herbsleb, 2018). Howison and Crowston (2014) investigated individuals' decisions to do certain tasks and found that a single individual performed the majority of tasks and that, when individuals deferred tasks, getting that task done required more than one person's expertise. Ke and Zhang (2010) identified the influence that various types of motivations have on how much effort FLOSS developers expend on tasks. Choi, Chengalur-Smith, and Nevo (2015) investigated the influence that three community markers (ideology, loyalty and identification) have on passive users'

behaviors, such as user brand extension, word-of-mouth, community involvement, and endorsement. Wen, Forman, and Graham (2013) showed how lawsuits influence user interest and developer activity in FLOSS software. Singh, Tan, and Youn (2011) investigated how individuals with different learning states learn from their peers versus from their own learning experience. Sojer and Henkel (2010) investigated factors that influence developers' code reuse decisions. They found that individuals with larger networks tend to reuse existing code more than other developers.

To sum up the six types of FLOSS research, the more macro-level FLOSS research, such as the country, industry-, market-, and company-level research, includes macro decisions about FLOSS such as decisions about adoption strategies, how to create value for companies, and how to best reward employees or price software. The decisions at the most micro level (namely, at the individual level) focus on individual decisions such as whether to participate or not, how much to commit to FLOSS, which tasks to take on, or whether to reuse code or not. Those studies that investigate project-level decision-making examine general governance in FLOSS teams, such as who has decision-making power. However, we lack empirical research that opens up the black box of FLOSS decision-making processes. We can readily see this gap in the fact that review and research framework papers on FLOSS have scarcely covered the topic (Aksulu & Wade, 2010; Crowston et al., 2012; Nelson et al., 2006; Niederman et al., 2006a; Scacchi, 2007; von Krogh & von Hippel, 2006).

2.2 Phase Theories of Team Decision-making Processes

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

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 a decision (e.g., by voting). Implicit consensus refers to occasions where one or more members make the decisions in a public forum, which means that all team members can observe the decision due to the openness that the ICT provides, but where others have not explicitly expressed agreement or disagreement. The idea of implicit consensus reflects the fact that, in FLOSS teams, communication and work rely on open (and, thus, transparent) broadcast media. Thus, team members implicitly agree to any teamwork that any member has shared and that none have rejected. Of course, apparent implicit consensus may also result from non-participation in the process, but repeated non-participants essentially cease to be team members, which means that implicit decisions still reflect a consensus of the active team participants.

In this section, we review prior theories on team decision-making processes as a basis for identifying decision-making process in FLOSS teams.

Researchers have proposed various frameworks to describe the phases in team decision-making processes. A phase refers to as "a period of coherent activity that serves some decision-related function, such as problem definition, orientation, solution development, or socio-emotional expression" (Poole & Baldwin, 1996, p. 216). Early studies proposed normative models to describe how teams make decisions in a unitary sequence of decision phases (Poole & Roth, 1989), which suggests that teams follow a systematic logic to reach decisions (Miller, 2008).

However, Poole (1983) and Poole and Roth (1989) have suggested that the normative models cannot adequately capture the dynamic nature of decision-making sequences and proposed another class of phase models, multiple-sequence models. 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 & Baldwin, 1996, p. 217). Based on studying 47 team decisions, Poole and Roth (1989a) 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; the team does not plan them ahead of time.

Multiple-sequence decision-making models 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, Raisinghani, & Theoret, 1976; Poole & Roth, 1989). Further, multiple sequence models provide guidance for practitioners to adapt to

changing demands (Poole, 1983; Poole & Baldwin, 1996) by providing a framework for structuring analyses of decision processes, terminology, and a basis for comparison between diverse processes. As such, we adopted this approach in this paper.

As a starting point for our analyses, we used the extant literature on sequence models and the studies that identify decision-making process phases based on team communications analyses (Mintzberg et al., 1976; Poole & Baldwin, 1996). Specifically, we adapted the decision functions coding system (DFCS) that Poole and Roth (1989) developed to the FLOSS context to identify different decision-making processes. We detail this system and our adaptations in Section 4.2.

3 Research Method

We turn now to how we designed a study to address our research question. Given the exploratory nature of our research, we designed a qualitative study. We collected 300 decision episodes from five FLOSS projects and analyzed their content to identify distinct decision-making processes.

3.1 Case Selection Decision to Control for Unwanted Systematic Variance

We sought to choose projects that would provide a meaningful basis for comparison across the three contextual factors. Given the diversity in FLOSS business models, to control for unwanted systematic variance, we chose community-based projects with internal governance structure that had a roughly similar age and a 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 means we could observe rich team interaction processes around development. Acknowledging that the development tools that such projects used might structure the decision-making processes, we selected projects on SourceForge (www.sourceforge.net), a FLOSS development site popular at the time we collected data that provides a consistent ICT infrastructure to developers. Table 1 summarizes the selected cases, which we describe further below.

Table 1. Project Comparison

Project name / category	Gaim[1] (Pidgin)	Fire	aMSN	WebERP[1]	OFBiz (Blackduck, n.d.)
Type	Instant messaging client	Instant messaging client	Instant messaging client	Enterprise resource planning (ERP) system	Enterprise resource planning (ERP) system
Lines of code	244,709			6,499,251	1,490,772
Mostly written In	C	C, C++, Objective C	Tcl/Tk	PHP	Java
Webpage	Pidgin.im	Fire.sourceforge.net	www.amsn-project.net	www.weberp.org	Ofbiz.apache.org
Type	Multi-Protocol	Multi-Protocol	Single-Protocol	N/A	N/A
Project license	GPL	GPL	GPL v2	GPL	Apache v2
Developers	10	12	41 (aMSN Messenger, n.d.)	27	35
Initial Release	November, 1998	April, 1999	May, 2002	January, 2003	November, 2001

We picked two different types of software. Specifically, we selected projects that developed instant messenger (IM) clients and enterprise resource planning (ERP) systems based on the expectation that these two types of projects would differ in complexity, which, in turn, would affect their decision-making processes. Due to the complexity of the ERP software, ERP developers tend to have in-depth knowledge of at least one software module. While they may understand how the module connects to other modules, they may not be experts in all aspects of the software. IM clients, on the other hand, may attract developers who may be more generalists.

We initially chose three cases for each project type: Gaim (currently known as Pidgin), aMSN, and Fire for the IM projects and Compiere, WebERP, and OFBiz (currently known as Apache OFBiz[2]) for the ERP

[1] We collected most of the data on Gaim (Pidgin), OfBiz, and WebERP from Openhub.net using the compare projects function.

projects. However, during data analysis, we came to realize that Compiere did not constitute a community-based project like the others since a company started it and it had both community and commercial aspects in its development. Therefore, one would better classify it as a team with governance toward outside parties based on de Laat's (2007) governance categorizations. To avoid possible bias due to this project, we decided to remove it from our study, which resulted in five (three IM and two ERP) projects in the final design.

ERP systems constitute one of the most complex software types (Parr, Shanks, & Darke, 1999; Sumner, 2000) for several reasons. First, a typical ERP system has many modules and features that are distributed across a company's different functions. For example, OFBiz has accounting (general ledger, accounts receivable, accounts payable, fixed assets), customer resource management, order management, e-commerce, warehousing and inventory, manufacturing, and material requirements planning (MRP) modules. Similarly, WebERP provides general ledger, accounts payable, accounts receivable modules, purchase/procurement module, inventory module, sales and order management module, customer relationship management module, supply chain-management module, document-management system module, payroll and attendance module, SMS and email module, and security module. OFBiz provides product and catalog management, promotion and pricing management, supply chain fulfillment, contracts, and payments and billing features spread between sales, marketing, customer management, supply chain, accounting, and finance. Each of these areas requires unique domain knowledge and technical knowledge. Rettig (2007) suggested that these systems have so much complexity that developing and changing them becomes risky because no single person in an organization could possibly know how a change in one part of the software will affect its functioning elsewhere (p. 22). Modules in the ERP software have high software code interdependencies and many external knowledge constraints such as accounting rules and legal reporting requirements. ERP software developers also need to consider how diverse companies can engineer the software to fit their diverse needs. Second, ERP systems integrate high volumes of data that either did not exist previously or one could not derive with other software (Chaudhari & Ghone, 2015). Further, the level of automation that ERP provides (Haddara, 2018) adds to the software's complexity. Glass (2003, p. 58) suggests that, for every 25 percent increase in complexity in the tasks one wants to automate, the software's complexity rises by 100 percent. Due to these factors, ERP systems—"massive programs with millions of lines of code, thousands of installation options, and countless interrelated pieces—have "introduced new levels of complexity" (Rettig, 2007, p. 23). Their complexity partly comes from their sheer size, which one can see in how many lines of code they have (1.5 million and 6.5 million for OFBiz and WebERP, respectively). A Carnegie Mellon Study found that the average professional coder makes 100 to 150 errors for every 1,000 lines of code (Mann, 2002), which means an ERP system such as WebERP with 6.5 million lines of codes could have anywhere between 650,000 to 975,000 bugs to fix as it undergoes development.

In contrast, IM clients have one main function and several features. Developers may need only purely experiential knowledge based on their using the IM software that they want to contribute to. IM clients may have only thousands of lines of code rather than the millions that ERP systems do. Further, many more individuals may participate in programming IM clients since it requires less skill and, thus, has lower entry barriers. Therefore, one can expect that the IM projects would have relatively simpler decision processes (e.g., they would require fewer individuals' input). To sum up, due to the differences in the type and variety of knowledge needed to develop ERP and IM software, we expect the decision-making processes in ERP projects to differ from those in IM projects.

4 Identifying the Patterns of Decision-making Process

In this section, we describe the research method we followed in each phase to identify different decision-making processes and the corresponding results.

4.1 Data and Unit of Analysis

We decided to initiate our investigation with a uniform communication and decision-making tool that exists across all FLOSS teams with which we could more easily apply and generalize our findings to other

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

similar yet non-FLOSS contexts. We focused on identifying generic decision-making processes for strategic and tactical decisions that we could test. In collecting strategic versus tactical decision episodes, we used the following definitions: we defined tactical decisions as decisions whose central issues relate to an indication for a change in the software code, such as accepting a patch or code that would become part of the code base. We defined strategic decisions as decisions whose central issues do not relate to code, such as legal issues, membership issues, funding, maintaining a positive group atmosphere, and software architecture.

Before we collected our data for this study, we followed various decision-making venues, such as issue trackers and projects' instant messaging tools such as Gaim and the developers' forums[3]. We observed that the developers' forums covered both strategic decision and tactical decisions the best, whereas developers focused solely on technical issues such as solving bugs or new features in issue trackers, and they typically[4] used instant messaging tools such as Internet Relay Chat (IRC) to ask for advice on an area that they had become stuck on rather than to make decisions at the group level. Our decision to collect data from a tool that hosts both strategic and tactical decision-making concurs with the phase-based decision-making theories that we use for this study. The decision-making literature includes both strategic and tactical decision-making processes: researchers conducted key studies that informed this literature stream with both strategic decision-making teams and student teams making tactical decisions (e.g., Poole, 1983).

We obtained data from the SourceForge website. In analyzing the interactions between developers on their forums, we did not find any references to offline discussions, which suggests that this data source provided a complete view of the decision-making process (at least for the decisions we analyzed for this study). Furthermore, we checked for and did not find evidence of discussions/decisions being split among different communication media when we specifically searched for issues across different media. Therefore, we could observe full decision episodes in the developers' forums.

As our primary unit of coding and analysis, we selected the decision episode, which we define as a sequence of messages that begins with a decision trigger that presents an opportunity or a problem that individuals need to decide on, that includes at least more than one message, and that possibly ends with a decision announcement (Annabi, Crowston, & Heckman, 2008). To give an example, a decision trigger may refer to a feature request or a report of a software bug. A decision announcement may refer to either a statement about one's intention to do something or about how one has actually implemented a fix. Note that some decision processes did not result in a decision that someone announced to the group, while others had multiple announcements individuals revised decisions. The messages in an episode capture the interactions among team members that constitute the process of making a particular decision from start to finish.

We identified decision episodes from the continuous stream of available messages through an initial coding process that the first and second authors performed independently. 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 organized discussions in a thread and occasionally initiated 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. In evaluating the threads in this explorative manner, we found that any such follow- typically appeared within the following month and, in more extreme cases, within three months. Therefore, we searched for messages on the same or similar content up to three months after the posting date of the last message in a thread. Since we analyzed 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 due to feedback from the team. The pairwise inter-coder reliability reached 85 percent and 80

[3] During our data collection, no project used GitHub.
[4] We observed that developers generally used IRC used to get quick help from fellow coders. However, we acknowledge that one anonymous reviewer noticed some decisions being made in IRC in their research. For this reason, researchers should be familiar with the practices that the community they research use.

percent on decision triggers and decision announcements, respectively. The coders reconciled all differences between through discussion to obtain the episode sample for analysis.

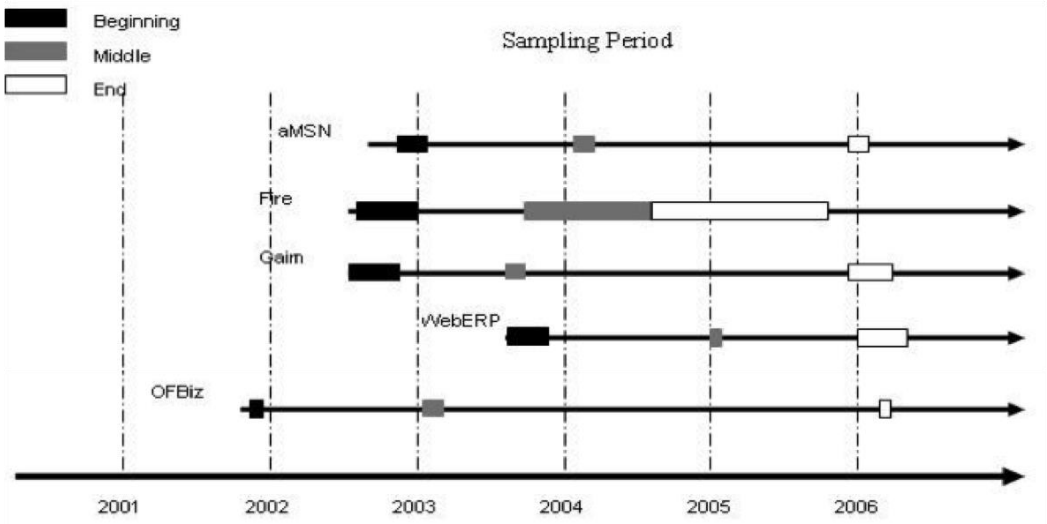

Figure 1. Sampling Periods for Decision Episodes by Project

In investigating decision-making processes, one needs to consider that the decision-making dynamics in community-based FLOSS projects develop over time due to the nature of participation among voluntary community members.(Benbya and Belbaly (2010) showed that both the type of participation and the level of effort that individuals make differ based on their motivation to gain knowledge in a specific area. Further, the type of individuals' participation to the decision-making process may change based on how much knowledge they have in an area. Accordingly, we stratified the decision episodes we identified by time: we chose 20 episodes from the beginning, middle, and end periods of each project[5] based on a concern that the decision-making process might differ in different software-development stages (e.g., initial collaboration vs. more established collaboration). 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. Figure 1 depicts the sampling periods for decision episodes by project.

Due to this initial coding process, we identified 300 decision episodes that each included messages with a trigger and (when present) one or more decision announcement(s). We chose the sample size to balance analysis feasibility with sufficient power for comparisons. With 60 episodes per project, we had reasonable power for comparison across projects while keeping the coding effort feasible.

4.2 Developing the Coding Scheme for Decision Processes

Once we obtained the decision episodes, we analyzed their content by coding the text segments that embodied the decision-making steps to identify the decision-making process in each episode. We deductively developed the coding scheme in two steps. First, as we note above, we adopted the decision functions coding system (DFCS) that Poole and Roth (1989) developed. As its primary unit of coding, this coding system uses the "functional move", which refers to "the function or purpose served by a particular segment of the conversational discourse" (Wittenbaum et al., 2004). Researchers have extensively used functional moves to understand the nature of interaction in both face-to-face and computer-mediated environments (Herring, 1996; Poole & Holmes, 1995; Poole, Seibold, & McPhee, 1985). However, few studies have used functional move to analyze complex, asynchronous, text-based environments such as email, bulletin boards, or threaded discussion forums. We used functional moves to identify the function of

[5] For each project, the beginning and the ending periods comprised the first and last 20 decision episodes, respectively, that we found at the time we collected the data (i.e., from the start of the project's online presence to the most recent period). The middle period for each project comprised 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 represents a key team decision in a FLOSS project.

messages in each episode. Note that a single message might include zero, one, or multiple functional moves.

In DFCS, functional moves for decision making include steps for problem analysis and 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. As a result, we produced an "IDEA" framework with four overall phases: decision identification (I), development (D), evaluation (E) and announcement (A). Each phase included one or more specific functional moves.

Second, we adapted the scheme to the FLOSS setting. To do so, we pilot coded 20 sample episodes and discussed how the scheme applied to the data. Due to these discussions, we removed the functional moves that seemed to not pertain to the FLOSS context from the coding scheme (such as "screening issues" and "authorizing decisions") and identified and added levels of detail that did uniquely pertain to the FLOSS content that we had not seen in previous studies. Using the revised scheme, we then coded a further 20 episodes and discussed the results until no new patterns emerged. We discuss the revision and the final revised coding scheme in the Appendix.

According to this coding scheme, the independent coders observed a perfectly rational decision-making process when the decision went through all four phases that the following sequential activities represent:

1) In the **identification (I)** stage, FLOSS 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).

2) In the **development (D)** stage, members may begin to discuss how they or software developers in the field generally resolve such problems (D-1). Team members either look for existing solutions (D-2) or try to design a specific solution for the problem (D-3).

3) In the **evaluation (E)** stage, team members evaluate the options they 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).

4) Finally, in the **announcement (A)** stage, a team member presents final team decision on how to solve the issue to the group (A-1).

Figure 2 shows how we coded these functional moves based on an example from the Gaim project. This process went through all four phases (I.E., identification, development, evaluation, and announcement) consecutively; however, we identified 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 occurred.

Once we established a coding scheme, two analysts independently coded the functional moves in the collected decision episodes and then compared their results. The initial coding revealed about 80 percent agreement. They discussed discrepancies until they fully agreed on each code. After they resolved all disagreements, they repeated coding until they fully agreed on all coded segments. This iterative coding process took about one month. The pairwise inter-coder reliability reached 85 percent and 80 percent, respectively, on decision triggers and decision announcements.

However, when analyzing process data at the most detailed level, processes can show great variability, which makes it hard to find theoretically meaningful patterns. To address this problem, we clustered the 300 coded decision-making episodes along two dimensions based on the sequences of moves represented in the episodes. The first dimension, coverage, refers to the extent that one observes theoretically identified decision-making phases in the public process. The second dimension, cyclicity, refers to whether the decision episodes progress linearly through the phases as in a normative model or loop through phases repeatedly as researchers such as Mintzberg et al. (1976) suggest. From here on, we refer to these two categories as linear and iterative decision-making processes, respectively.

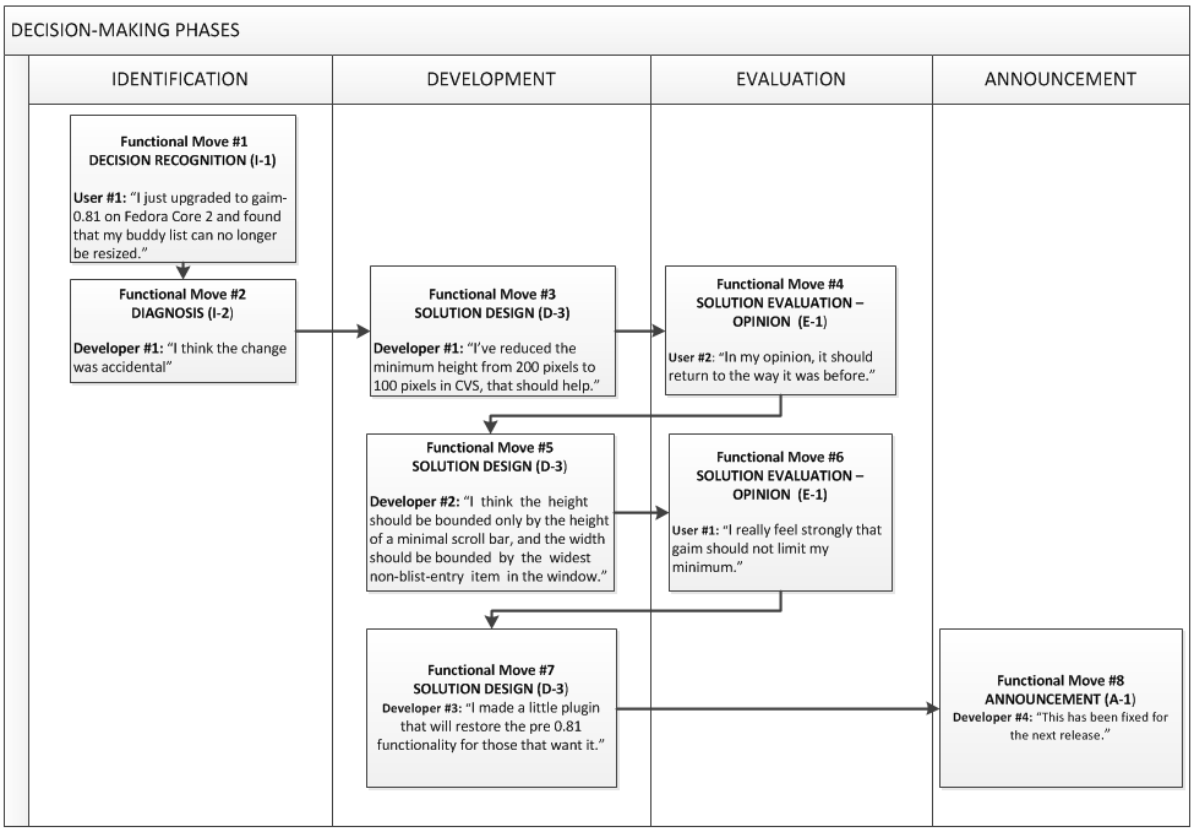

Figure 2. An Example Illustrating How We Coded a Decision for Functional Moves

4.3 Findings: Qualitative Analysis of Decision-making Patterns

Following the procedure that we describe in Section 4.2, we sorted the 300 decision-making episodes into five clusters according to the number of phases. We labeled these processes shortcut, implicit development, implicit evaluation, complete, and abandoned (i.e., lacking a final decision announcement). We depict each process's pattern in Figure 3. The dashed lines in the figures indicate points with possible loops that lead to iterative decision process. The loop from decision announcement to previous phases indicates that team members announce one or more intermediate decisions before they finalize the decision.

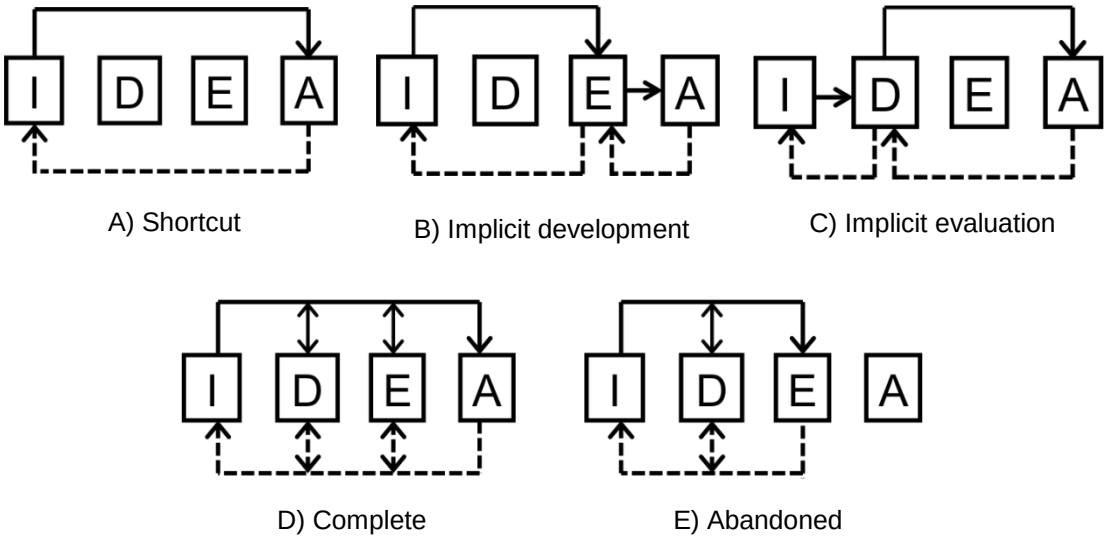

Figure 3. Five Decision-making Processes Identified based on the Data

4.3.1 Shortcut

This process represents the simplest pattern in which team members make a decision right after recognizing an opportunity and possibly briefly diagnosing a problem. It does not explicitly include any solution development or evaluation. We observed this process 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) after which an administrator quickly responded that he "just fixed it" (A-1, decision announcement) with no further discussion or evaluation. While this process lacks team input, we argue that these decisions still represent 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., it has consequences for the team).

4.3.2 Implicit Development

In this process, team members skip the solution-development phase, which does not mean that they did not develop a solution but rather that we did not find 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 provided 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 a message 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 text did not visibly contain the solution-analysis, search, and design steps. However, at least the user who sent the patch (and possibly others who did not feel it necessary to report their progress) conducted them.

4.3.3 Implicit Evaluation

In this process, a team member announces a decision directly after the individual generates solution alternatives without explicitly evaluating the alternatives. The way team members jump from alternatives to a solution based on one of the alternatives indicates that the team member evaluates and picks the best alternative by doing an evaluation in their mind without making the evaluation process explicit to the reader. 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 concerned the aMSN project or its supporting software (e.g., a KDE problem) (I-2, diagnosis). After some discussion and testing, members confirmed it constituted an aMSN tray icon problem (I-2, diagnosis). They then turned its attention to suggesting alternative solutions (D-3, solution design) and quickly fixed the problem (A-1, decision announcement).

4.3.4 Complete

In this process, the team goes through all decision-making phases 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 process most closely resembles the rational approach that earlier studies have described. 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 solution's usability (E-2, solution evaluation—action). Subsequently, the administrator promised to commit the code into CVS soon (A-1, decision announcement).

We observed iterative processes with more complex issues. These issues' complexity stems from the fact one cannot diagnose and resolve them without also addressing other subissues. As the subissues relate to one another, discussions may loop back to any previous phase at any time. We sometimes found another trigger that we could have interpreted as starting a new decision episode in these issues. However, since the issues relate to one another, it would not be faithful to the original issue's source 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. He first asked: "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. In evaluating it, he indicated it may be an easily changeable 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, a user then jumped in to recommend an alternate solution, which took 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, which led to several development and evaluation loops before they agreed on a solution.

4.3.5 Abandoned

In this process, team members do not announce a decision. Abandonments may occur in any phase of discussion and happen for various reasons. Team members may abandon a decision-making process during the identification phase due to a disagreement on whether a real problem exists or whether they need to fix a problem that does exist. They may abandon it during the development phase due to disagreement about the merits of different technical approaches and concerns. They may abandon it in the evaluation phase due to following different 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: 1) releasing a stable version with minor changes or 2) releasing an unstable version with a major innovation (D-1, solution analysis). Both sides extensively examined the current solutions, considered relevant consequences, 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 2 shows the distribution of the five decision-making processes across the 300 decision episodes. From the table, we can see that only 38 percent of the decisions episodes we analyzed went through all four phases (i.e., "complete"), while 52 percent of the discussions reached a decision while skipping one or two phases (shortcut, implicit development, or implicit evaluation). Team members reached no decision in the remaining 10 percent of cases (abandoned). In 23 percent of the decision episodes, team members decided right after they recognized the decision trigger (shortcut process). While team members made 28 percent of decisions without the evaluation phase (implicit evaluation), they made only one percent of the decisions without a visible development phase (implicit development).

Table 2. Count of Observed Decision-making Processes for All Episodes

	Short-cut	Implicit development	Implicit evaluation	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%)**

When we looked for differences in the patterns that the ERP and IM projects exhibited, we did not observe any systematic difference in the decision processes. χ^2 tests[6] (see Tables 3 and 4) showed similar patterns in how the two project types used different decision processes for both tactical decisions (χ^2 =1.644, p = 0.649) and strategic decisions (χ^2 = 6.521, p = 0.100). The different types of knowledge that the ERP FLOSS developers required did not seem to cause them to use different phases or functional moves than those that we explain above and provide in the coding scheme in the Appendix.

[6] Since we found only four cases of implicit development episodes, we excluded them from this analysis.

Therefore, we can conclude that the type and extent of knowledge required for software does not influence the decision processes that FLOSS development teams use.

Table 3. Distribution of Decision-making Processes between IM and ERP Projects for Tactical Decisions

	Shortcut	Implicit evaluation	Completed	Abandoned	Total
IM	37 (27%)	44 (32%)	45 (32%)	13 (9%)	139 (100%)
ERP	17 (22%)	27 (34%)	30 (38%)	5 (6%)	79 (100%)
Total	**54 (25%)**	**71 (33%)**	**75 (34%)**	**18 (8%)**	**218 (100%)**
$\chi^2 = 1.644$, df = 3, p = 0.649					

Table 4. Distribution of Decision-making Processes between IM and ERP Projects for Strategic Decisions

	Shortcut	Implicit evaluation	Completed	Abandoned	Total
IM	6 (16%)	4 (11%)	24 (63%)	4 (10%)	38 (100%)
ERP	10 (25%)	8 (20%)	14 (35%)	8 (20%)	40 (100%)
Total	**16 (21%)**	**12 (15%)**	**38 (49%)**	**12 (15%)**	**78 (100%)**
$\chi^2 = 6.251$, df = 3, p = 0.100					

Lastly, we clustered the decision-making processes based on the cyclicity. We found that 39 percent of decisions followed a linear decision process, while the other 61 percent included one or more loop backs and followed an iterative decision process.

5 Discussing Findings and Theoretical Contributions: Multiple Sequences of Decision-making Processes in FLOSS Development

In this study, we investigated decision-making process due to IS research's focus on how information technologies influence and support decision processes (Huber, 1990; Shaikh & Karjaluoto, 2015). Information technologies do influence and support decision processes, especially for community-based FLOSS projects with internal governance in which team members make decisions virtually, asynchronously, across different time zones, and almost exclusively via information systems (Crowston et al., 2012). Enabling and supporting virtual, asynchronous decision-making that spans different geographical locations and time zones requires one to understand the decision-making processes that we identify. Only then can one identify the right types of new information technologies that support the processes. Research on group support systems—a subset of research on decision support systems that focuses on developing information systems that support the decision-making process in groups—receives significant funds from both grants and industry (Arnott, Pervan, & Dodson, 2005). Such funding indicates the ongoing need for research to understand various teams' decision-making processes so that well-designed group support systems can be developed to support these decision-making processes. To support this point, Watson (2018) notes that "decision support systems should enable and boost interdependent decision making, which involves groups of people and should support all phases of the decision-making process, intelligence, design, and choice" (p. 375). While Watson refers to three phases (i.e., intelligence, design, and choice), we contribute to the group support systems research by finding that FLOSS teams differ in that they use four phases for decision making that incorporate the development stage (solution development) and evaluation stage (evaluation of the developed solutions). Further, we identified noticeable sequences in the FLOSS decision-making process such as phase skipping and iteration back to earlier phases, which we describe in more detail below.

We contribute to the decision-making literature predominantly by identifying five different decision-making processes that we observed in community-based FLOSS development teams with internal governance. In identifying these processes, we help fill in the gap in the literature that Nelson et al. (2006) identified on the lack of investigations into the extent to which one can explain FLOSS decision mechanisms using classical theories from organizational structure. The extant research on FLOSS has investigated various decisions related to the strategic aspects that influence FLOSS developers such as the FLOSS strategy (Pykäläinen et al., 2009), FLOSS adoption decision and innovation with FLOSS (Maldonado, 2010), decisions on hybrid business models that include FLOSS (Deodhar et al., 2012), company-level decisions

such as strategic decisions (Alspaugh et al., 2010), FLOSS acquisition/adoption decisions (Benlian, 2011; Chengalur-Smith et al., 2010; Marsan et al., 2012), licensing decisions (Singh & Phelps, 2013), employment contract decisions (Mehra et al., 2011; Mehra & Mookerjee, 2012), and pricing decisions (August et al., 2013; Machado et al., 2017). Similarly, the decisions that research has investigated at the project level, such as the decisions that contribute to the FLOSS development processes (e.g., Howison & Crowston, 2014; Wang et al., 2014; Wei et al., 2014) and FLOSS leadership process (Eseryel & Eseryel, 2013), do not explicate the decisions sufficiently enough to identify decision-making elements that one can then support with group decision-making technologies. Lastly, the individual-level FLOSS research focuses more on the elements that support individual contribution decisions rather than the decision-making process itself. The investigated factors that contribute to individual participation decisions include motivation (Benbya & Belbaly, 2010; Ke & Zhang, 2010; von Krogh et al., 2012), commitment (Bateman et al., 2011; Daniel, Maruping, et al., 2018), task selection (Howison & Crowston, 2014), community markers (Choi et al., 2015), and lawsuits related to the FLOSS (Wen et al., 2013). This research stream does not investigate what decision processes individuals go through after they make the decision to participate in FLOSS. Since we investigate such processes, we contribute to the FLOSS literature at the individual level.

We developed two sets of insights from our analysis regarding 1) decision processes and 2) patterns in which individuals used these processes. We saw decision-making processes in community-based FLOSS development with internal governance as having multiple sequences that reflect the FLOSS setting's unique characteristics. In this research, we identified five different decision-making processes whose phases varied in both number and sequence: shortcut, implicit development, implicit evaluation, complete, and abandoned. We observed four patterns in how FLOSS team members used these processes: frequent shortcuts, frequent implicit evaluation, infrequent implicit development, and many cycles looping back to previous decision-making stages. We explain these patterns based on 1) FLOSS development's unique characteristics and 2) FLOSS decision processes' high dependence on information technologies.

First, we observed that the decision-making processes as exhibited in the discussion forums differ from those that researchers have observed in other decision-making contexts. For example, Mintzberg et al. (1976) argue that any decision process must include the evaluation-choice of a solution (evaluation in our case). However, in our study, team members made 23 percent of the decisions without explicitly discussing solutions at all (i.e., 70 of 300 decisions were shortcut). At first, one may find the high frequency of shortcut decisions in what many often describe as an open and participative setting surprising. In addition to shortcut decisions, we found that 28 percent of decision episodes (83 out of 300) followed the implicit evaluation process that skips the evaluation phase. In contrast, only one percent (4 out of 300) followed the implicit development process, which includes an evaluation phase but skips the development phase.

At first, these results seem paradoxical: open projects that make decisions in a seemingly opaque and non-participatory fashion. Our finding that they commonly rely on shortcut processes—an individual decision-making process—to make decisions that bind the team as a whole is unique to the group decision-making literature. While we cannot completely rule out the existence of unarchived offline discussion that contains the missing phases, it appears that the lack of the evaluation phase and other decision-making phases reflects an action orientation for decision making in FLOSS development teams (Eseryel & Eseryel, 2013): that it is preferable to simply try out a solution rather than performing evaluating potential alternatives in detail in advance. We can see this value in the following description of the Internet Engineering Task Force's decision process): "We reject: kings, presidents and voting. We believe in: rough consensus and running code" (Clark, 1992, p. 543). As a result, FLOSS teams focus on decision processes that emphasize making a sufficiently good decision based on as much collaboration as needed rather than spending too much time evaluating options to find a perfect solution with complete collaboration.

Second, the missing phases may also provide empirical support for stigmergic coordination in FLOSS development (Bolici, Howison, & Crowston, 2015). By examining those decision episodes using simpler decision processes, we found that many had mentioned or referred to software codes explicitly in their discussion. Prior research has proposed that stigmergic coordination makes explicit discussion unnecessary (Crowston, Østerlund, Howison, & Bolici, 2011; Robles, Merelo, & Gonzalez-Barahona, 2005). Namely, the information artifact's (i.e., the software code's), shared and transparent nature and the technical ability to reverse code submissions when disagreements arise enable FLOSS teams to rely on shortcut decision-making processes. The shortcut decision-making process constitutes a valuable process

for group decision-making when combined with the IT infrastructure mentioned above in that it eliminates (the cost of) unnecessary communication and coordination. While researchers have previously discussed the idea of stigmergic coordination (Crowston et al., 2011; Robles et al., 2005), they have conducted no empirical research that examines how stigmergic coordination of decision making takes place. 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 effectively participate.

Moreover, we found that developers often raised questions about others' actions based on their knowledge, which led back to previous decision-making phases and resulted in a high proportion of cyclic processes (181 out of 300 or 61%). While our findings concur with the observation that "IS decisions are often complex and dynamic" (Boonstra, 2003, p. 206), the factors that researchers have previously used to explain this cyclicity, such as political influences, urgency, and necessity (Eisenhardt & Zbaracki, 1992; Mintzberg et al., 1976), 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 information technologies that allow for asynchronous communication and collaboration, which means that anyone can observe and contribute to a decision in process and even later join a discussion that has occurred for some time. This pattern may also reflect the fact that no individual organizes the discussions to follow a normative path as one would observe in teams with managers or decision support systems to structure the decision process.

While, in organizational settings, decision making's dynamic nature may to an extent indicate inefficiencies, in an open setting such as FLOSS where voluntarily developers develop software, the process allows participants the opportunity to jump in at any time to contribute to work and related decisions, which increases the level of cyclicity in FLOSS decision making. In conclusion, we suggest that the cyclicity in these teams results from their self-organizing nature and their use of asynchronous communication media rather than the factors that researchers suggested to lead to cycles (such as political factors) in other decision-making teams.

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

We need to identify these processes because the decision process that a group uses directly affects its performance. As such, in examining the microstructures of decision-making processes in depth, we complement existing macro-level research on decision making (e.g., German, 2003; Raymond, 1998). The frequency and type of decision-making processes that FLOSS teams use can be inputs for future theory-development efforts that predict group performance. For example, quantitative studies could compare the types of decision processes that groups use and their effectiveness (or overall project success).

Lastly, based on the literature, we had expected that FLOSS teams that develop software that require many different types of external and internal knowledge to use different decision processes than those that develop software with more generic knowledge requirements. This expectation influenced our case-selection strategy. However, contrary to our expectation, we did not observe differences in the decision-making processes that simple (IM) and complex (ERP) software projects used. Thus, our findings suggest that FLOSS projects tend to adopt similar decision-making processes for decisions despite the complexity and the knowledge requirements of the software that they develop. This similarity reflects the observation that the projects seemed to organize the software-development process in a similar way: they used the same ICT tools in discussion and implementation spaces, parallel development, and debugging that involved loosely centralized and gratis contributions from individual voluntary developers (Feller & Fitzgerald, 2000), which resulted in developers selecting problems with a similar scope and similar decision demands. This finding suggests that one can generalize the decision processes we identified across the whole spectrum of community-based FLOSS projects with internal governance and perhaps to other kinds of FLOSS as well.

5.1 Limitations and Future Research

At the beginning of this study, we highlighted that various types of FLOSS teams with difference governance mechanisms exist. We specifically focused on FLOSS teams that have internal governance, which we call "community-based FLOSS teams". These projects have existed for a period of time, have

multiple participants, feature internal governance, and require coordination and control to achieve desired outcomes. Therefore, researchers should test our findings for FLOSS teams that may have different governance types to see if they one can extend them to these teams. Specifically, relatively new FLOSS teams and highly institutionalized teams (which occur when a non-profit foundation protects a project or when a project works with companies) may show different decision-making dynamics. Therefore, researchers should test the decision-making processes we identified in these two types of FLOSS teams to see if they generalize to those settings.

Second, we limited our investigation into decision-making processes to discussion forums. By using this approach, we could capture both strategic and tactical decision-making processes. However, different communication media may provide different affordances (Volkoff & Strong, 2013). Therefore, future research should test the five decision-making processes that we identified in different types of communication media, such as issue trackers or pull requests that FLOSS teams use. The numerous communication media that FLOSS teams use include various automated listservs that automatically send emails whenever someone commits a new patch, specialized listservs for individuals working on translations, and team websites or wikis. Further, issue trackers help coordinate decisions on technical issues such as bug reports or enhancement requests. Some FLOSS teams use GitHub, which enables them to make pull requests to create various changes on a branch. Teams may use pull requests to discuss, review, and edit various changes that members do on a commit before they finalize these changes and commit them to the base branch. These pull requests also create opportunities for interaction and decision making on a subset of a project.

Since the different communications tools that we mention above have different features, each tool may provide different affordances, which means that different communication media may offer different possibilities for action to users (Volkoff & Strong, 2013). Some only inform the members of the progress and, therefore, do not include the full interaction needed for team-level decision making, whereas others, such as issue trackers, focus only on technical decisions and have a unique structure that forces users to fill in different fields and, therefore, may affect the organization of the decision-making process. GitHub tools may better suit those people who focus on a subset of the project, such as developing the website or a specific branch. We expect to see similar decision-making processes across different media because we investigated social practices that information and communication technologies supported. Yet, to the extent different communication and coordination media provide different affordances (Volkoff & Strong, 2013) related to decision making, they may show slight differences; therefore, researchers should test the decision-making processes we identified across different media.

As with any study, ours has several limitations. First, we excluded synchronous discussion media, such as IRC, instant messaging, or phone calls. However, we did follow the IRC and instant messaging channels for the IM projects in particular, such as Gaim, before we made the decision to focus on the developers' forums. In observing these channels, we found that individuals typically used them to clarify programming questions quickly rather than to make decisions. Thus, we decided to focus on developer forums to investigate community-level decision making.

Second, we had no way to observe one-to-one IM conversations that occurred outside the publicly shared ones. Thus, we want to acknowledge that a subgroup possibly conducted some steps in the decision-process that we infrequently observed using such alternative channels. Future research should consider the impact of communications synchronous communication channels on community-level decision making on developers' forums. However, we argue that, had individuals used such channels, it would not have changed our main conclusion that FLOSS team members make many decisions that bind their team to a course of action without the entire team's explicit involvement in seemingly important decision-making process phases.

Finally, our small sample size limits our findings (i.e., five projects and 300 decision episodes). While the small sample size meant we could conduct manual coding and obtain rich data to better understand 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 and ERP projects), which limits our results' generalizability. We specifically focused on these two projects because they represent two extremes on the continuum for the knowledge that they require: while ERP projects require unique domain knowledge in many areas (such as accounting, finance, marketing etc.) in addition to technical knowledge on these areas, IM projects require focus on one area, which many developers experientially have as software users.

Nevertheless, the decision processes and relationships we identify provide the foundation for deeper explorations into and potentially richer explanations of decision-making processes in FLOSS teams. Future research should apply our framework to a larger and more representative sample of FLOSS projects.

6 Practical Implications

Three groups of individuals in the practitioner community can benefit from our results: 1) participants and leaders of community-based FLOSS teams, 2) managers and members of companies who would like to actively contribute to existing community-based FLOSS teams or to develop and support such teams with independent internal governance, and 3) individuals who would like to bring to their organizations the FLOSS work model where internally governed small communities, such as the ones we investigated in this study, collaborate on technical projects.

Understanding decision-making processes also enables FLOSS teams to develop group decision support systems and other information systems that would fulfill the team requirements. For instance, if FLOSS team members use applications such as Algorithmic Autoregulation (Fabbri et al., 2014) and habitually record their coding processes as they do their work, they could better explicate the individual decision-processes that make up the shortcut decisions. To succeed, distributed teams such as FLOSS teams particularly require discussion and implementation spaces, which depend on such systems to both accomplish tasks and maintain the group.

Second, the success of FLOSS development has attracted more and more companies to participate in it (Dahlander & Magnusson, 2008). Companies first need to understand how FLOSS communities operate before they can successfully participate in FLOSS development. By understanding the decision-making processes in FLOSS teams, firms could better understand the decision processes that team members would likely use for different task types and, thus, adjust their behaviors to better contribute to FLOSS development.

Third, though we studied decision-making processes in FLOSS development teams, one can apply many of our findings to self-organizing organizational virtual teams and similar open organizations more generally. Indeed, Markus, Manville, and Agres (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).

Our results 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 information and communication technologies that enable team members to coordinate through their work product and augment these information and communication technologies with discussion tools in a way that mirrors FLOSS practices. For example, co-workers may be able to substitute examining shared documents (e.g., with tools such as Google Documents or Lotus Notes) for extensively discussing their content 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, FLOSS development's apparent advantages may become more widely available.

Acknowledgments

We thank Dr. Robert Heckman and Ms. Qing Li for their contributions to earlier versions of this article, which were published as conference proceedings.

References

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

Allen, J. P. (2012). Democratizing business software: Small business ecosystems for open source applications. *Communications of the Association for Information Systems*, *30*, 483-496.

Almarzouq, M. (2005). Open source: Concepts, benefits, and challenges. *Communications of the Association for Information Systems*, *16*, 756-784.

Alspaugh, T. A., Scacchi, W., & Asuncion, H. U. (2010). Software licenses in context: The challenge of heterogeneously-licensed systems. *Journal of the Association for Information Systems*, *11*(11/12), 730-755.

aMSN Messenger. (n.d.). *Current developers.* Retrieved from www.amsn-project.net/current-developers.php

Andriole, S. J. (2012). Seven indisputable technology trends that will define 2015. *Communications of the Association for Information Systems*, *30*, 61-72.

Annabi, H., Crowston, K., & Heckman, R. (2008). Depicting what really matters: Using episodes to study latent phenomenon. In *Proceedings of the International Conference on Information Systems.*

Arnott, D., Pervan, G., & Dodson, G. (2005). Who Pays for decision support systems research? Review, directions, and issues. *Communications of the Association for Information Systems*, *16*, 356-380.

August, T., Shin, H., & Tunca, T. I. (2013). Licensing and competition for services in open source software. *Information Systems Research*, *24*(4), 1068-1086.

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

Barrett, M., Heracleous, L., & Walsham, G. (2013). A rhetorical approach to IT diffusion: Reconceptualizing the ideology-framing relationship in computerization movements. *MIS Quarterly*, *37*(1), 201-220.

Bateman, P. J., Gray, P., & Butler, B. S. (2011). The impact of community commitment on participation in online communities. *Information Systems Research*, *22*(4), 841-854.

Benbya, H., & Belbaly, N. (2010). Understanding developers' motives in open source projects: A multi-theoretical framework. *Communications of the Association for Information Systems*, *27*, 589-610.

Benlian, A. (2011). Is traditional, open-source, or on-demand first choice? Developing an AHP-based framework for the comparison of different software models in office suites selection. *European Journal of Information Systems*, *20*(5), 542-559.

Blackduck. (n.d.). *Apache OFBiz trunk.* Retrieved from https://www.openhub.net/p/Apache-OFBiz

Bolici, F., Howison, J., & Crowston, K. (2015). Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms. *Cognitive Systems Research*, 38, 14-22.

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

Chaudhari, S., & Ghone, A. (2015). *ERP software market by deployment and function*. High Tech, Enterprise & Consumer IT.

Chengalur-Smith, I., Sidorova, A., & Daniel, S. L. (2010). Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, *11*(11/12), 657-683.

Choi, N., Chengalur-Smith, I., & Nevo, S. (2015). Loyalty, ideology, and identification: An empirical study of the attitudes and behaviors of passive users of open source software. *Journal of the Association for Information Systems*, *16*(8), 674-706.

Chua, C. E. H., & Yeow, A. Y. K. (2010). Artifacts, actors, and interactions in the cross-project coordination practices of open-source communities. *Journal of Strategic Information Systems*, *11*(12), 838-867.

Clark, D. D. (1992). A cloudy crystal ball: Visions of the future. In *Proceedings of the 24th Internet Engineering Task Force*.

Colombo, M. G., Piva, E., & Rossi-Lamastra, C. (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., Østerlund, C., Howison, J., & Bolici, F. (2011). Work as coordination and coordination as work: A process perspective on FLOSS development projects. In *Proceedings of the 3rd International Symposium on Process Organization Studies.*

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/libre open source software development: What we know and what we do not know. ACM Computing Surveys, *44*(2), 7:1-7:35.

Dahlander, L., & Magnusson, M. (2008). How do firms make use of open source communities? *Long Range Planning*, *41*(6), 629-649.

Daniel, S., Midha, V., Bhattacherjee, A., & Singh, S. P. (2018). Sourcing knowledge in open source software projects: The impacts T of internal and external social capital on project success. *Journal of Strategic Information Systems*, *27*(3), 237-256.

Daniel, S., & Stewart, K. (2016). Open source project success: Resource access, flow, and integration. *Journal of Strategic Information Systems*, *25*, 159-176.

Daniel, S. L., Maruping, L. M., Cataldo, M., & Herbsleb, J. (2018). The impact of ideology misfit on open source software communities and companies. *MIS Quarterly*, *42*(4), 1069-1096.

de Laat, P. B. (2007). Governance of open source software: State of the art. *Journal of Management and Governance*, *11*(2), 165-177.

Deodhar, S. J., Saxena, K. B. C., Gupta, R. K., & Ruohonen, M. (2012). Strategies for software-based hybrid business models. *Journal of Strategic Information Systems*, *21*(4), 274-294.

Di Tullio, D., & Staples, D. S. (2013). The governance and control of open source software projects. *Journal of Management Information Systems*, *30*(3), 49-80.

Eisenhardt, K. M., & Zbaracki, M. J. (1992). Strategic decision making. *Strategic Management Journal*, *13*(52), 17-37.

Eseryel, U. Y., & Eseryel, D. (2013). Action-embedded transformational leadership in self-managing global information technology teams. *The Journal of Strategic Information Systems*, 22(2), 103-120.

Fabbri, R., Fabbri, R., Vieira, V., Penalva, D., Shiga, D., Mendonca, M., Negrao, A., Zambianchi, L., & Thumé, G. S. (2014). The algorithmic autoregulation software development methodology. *Revista Eletrônica de Sistemas de Informaç*ão, *12*(3), 1-16.

Feller, J., Finnegan, P., & Nilsson, O. (2011). Open innovation and public administration: Transformational typologies and business model impacts. *European Journal of Information Systems*, *20*(3), 358-374.

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. In *Proceedings of the 21st International Conference on Information Systems.*

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587-598.

Gacek, C., & Arief, B. (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.

Glass, R. L. (2003). *Facts and fallacies of software engineering.* Boston, MA: Pearson.

Guzzo, R. A., & Salas, E. (1995). *Team effectiveness and decision making in organizations*. San Francisco, CA: Jossey-Bass.

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

Haddara, M. (2018). ERP systems selection in multinational enterprises: A practical guide. International *Journal of Information Systems and Project Management*, *6*(1), 43-57.

Herring, S. C. (Ed.) (1996). *Computer-mediated communication: Linguistic, social, and cross-cultural perspectives.* Philadelphia: John Benjamins.

Howison, J., & Crowston, K. (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.

Ke, W., & Zhang, P. (2010). The effects of extrinsic motivations and satisfaction in open source software development. *Journal of the Association for Information Systems*, *11*(12), 784-808.

Keen, P. G. W., & Cummins, J. J. (1994). *Networks in action.* Belmont, CA: Wadsworth.

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

Kiesler, S., & Sproull, L. (1992). Group decision making and communication technology. *Organizational Behavior and Human Decision Processes*, *52*(1), 96-123.

Love, J., & Hirschheim, R. (2017). Crowdsourcing of information systems research. *European Journal of Information Systems*, *26*(3), 315-332.

Machado, F. S., Raghu, T. S., Sainam, P., & Sinha, R. (2017). Software piracy in the presence of open source alternatives. *Journal of the Association for Information Systems*, *18*(1), 1-21.

Macredie, R. D., & Mijinyawa, K. (2011). A theory-grounded framework of open source software adoption in SMEs. *European Journal of Information Systems*, *20*(2), 237-250.

Maldonado, E. (2010). Process of introducing FLOSS in the public administration: The case of Venezuela. *Journal of the Association for Information Systems*, *11*(11), 756-783.

Mann, C. (2002). Why software is so bad. *Technology Review*. Retrieved from https://www.technologyreview.com/s/401594/why-software-is-so-bad/

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

Marsan, J., Pare, G., & Beaudry, A. (2012). Adoption of open source software in organizations: A socio-cognitive perspective. *Journal of Strategic Information Systems*, *21*(4), 257-273.

Mehra, A., Dewan, R., & Freimer, M. (2011). Firms as incubators of open-source software. *Information Systems Research*, *22*(1), 22-38.

Mehra, A., & Mookerjee, V. (2012). Human capital development for programmers using open source software. *MIS Quarterly*, *36*(1), 107-122.

Midha, V., & Bhattacherjee, A. (2012). Governance practices and software maintenance: A study of open source projects. *Decision Support Systems*, *54*(1), 23-32.

Miller, K. (2008). *Organizational communication: Approaches and processes.* Bonston, MA: Wadsworth Cengage Learning.

Mintzberg, H., Raisinghani, D., & Theoret, A. (1976). The structure of “unstructured” decision process. *Administrative Science Quarterly*, *21*(2), 246-275.

Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of Linux kernel. First Monday, 5(11).

Morgan, L., Feller, J., & Finnegan, P. (2013). Exploring value networks: Theorising the creation and capture of value with open source software. *European Journal of Information Systems*, *22*(5), 569-588.

Morgan, L., & Finnegan, P. (2014). Beyond free software: An exploration of the business value of strategic open source. *Journal of Strategic Information Systems*, *23*(3), 226-238.

Nelson, M., Sen, R., & Subramaniam, C. (2006). Understanding open source software: A research classification framework. *Communications of the Association for Information Systems*, *17*, 266-287.

Niederman, F., Davis, A., Greiner, M. E., Wynn, D., & York, P. T. (2006a). A research agenda for studying open source I: A multi-level framework. *Communications of Association for Information Systems, 18,* 129-149.

Niederman, F., Davis, A., Greiner, M. E., Wynn, D., & York, P. T. (2006b). Research agenda for studying open source II: View through the lens of referent discipline theories. *Communications of the Association for Information Systems*, *18*, 150-175.

O'Mahony, S., & Ferraro, F. (2004a). The emergence of governance in an open source community. *Academy of Management Journal*, *50*(5), 1079-1106.

O'Mahony, S., & Ferraro, F. (2004b). *Hacking alone? The effects of online and offline participation on open source community leadership.* Retrieved from https://bit.ly/2Sh6743

O'Mahony, S., & Ferraro, F. (2007). The emergence of governance in an open source community. *The Academy of Management Journal*, *50*(5), 1079-1106.

Parr, A. N., Shanks, G., & Darke, P. (1999). Identification of necessary factors for successful implementation of ERP systems. In O. Ngwerryama, L. Introna, M. Myers, & J. DeGross (Eds.), *New information technologies in organizational processes: Field studies and theoretical reflections on the future of work*. London, UK: Kluwer Academic Publishers.

Peng, G., & Dey, D. (2013). A dynamic view of the impact of network structure on technology adoption: The case of OSS development. *Information Systems Research*, *24*(4), 1087-1099.

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., & Baldwin, C. L. (1996). Developmental processes in group decision making. In R. Y. Hirokawa & M. S. Poole (Eds.), *Communication and group decision making* (pp. 215-241). Thousand Oaks, CA: Sage.

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

Poole, M. S., & Roth, J. (1989). Decision development in small group IV: A typology of group decision paths. *Human Communication Research*, *15*(3), 323-356.

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

Pykäläinen, T., Yang, D., & Fang, T. (2009). Alleviating piracy through open source strategy: An exploratory study of business software firms in China. *Journal of Strategic Information Systems*, *18*(4), 165-177.

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.

Rettig, C. (2007). The trouble with enterprise software. *MIT Sloan Management Review*, *49*(1), 22-27.

Robles, G. (2004). A software engineering approach to libre software. In R. A. Gehring & B. Lutterbeck (Eds.), *Open source jahrbuch*. Berlin: Lehmanns Media.

Robles, G., Merelo, J. J., & Gonzalez-Barahona, J. M. (2005). Self-organized development in libre software: A model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling.*

Santos, C., Kuk, G., Kon, F., & Pearson, J. (2013). The attraction of contributors in free and open source software projects. *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.

Setia, P., Rajogopalan, B., & Sambamurthy, V. (2012). How peripheral developers contribute to open-source software development. *Information Systems Research*, *23*(1), 144-163.

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

Singh, P. V., & Phelps, C. (2013). Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research*, *24*(3), 539-560.

Singh, P. V., Tan, Y., & Mookerjee, V. (2011). Network effects: The influence of structural capital on open source project success. *MIS Quarterly*, *35*(4), 813-829.

Singh, P. V., Tan, Y., & Youn, N. (2011). A hidden Markov model of developer learning dynamics in open source software projects. *Information Systems Research*, *22*(4), 790-807.

Sojer, M., & Henkel, J. (2010). Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems*, *11*(12), 868-901.

Sumner, M. (2000). Risk factors in enterprise-wide/ERP projects. *Journal of Information Technology*, *15*(4), 317-327.

Ven, K., & Verelst, J. (2011). An empirical investigation into the assimilation of open source server software. *Communications of the Association for Information Systems*, *28*, 117-140.

Volkoff, O., & Strong, D. M. (2013). Critical realism and affordances: Theorizing IT-associated organizational change processes. MIS *Quarterly*, *37*(3), 819-834.

von Krogh, G. (2009). Individualist and collectivist perspectives on knowledge in organizations: Implications for information systems research. *Journal of Strategic Information Systems*, *18*(3), 119-129.

von Krogh, G., Haefliger, S., Spaeth, S., & Wallin, M. W. (2012). Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, *36*(2), 649-676.

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

Wang, X., Kuzmickaja, I., Stol, K.-J., Abrahamsson, P., & Fitzgerald, B. (2014). Microblogging in open source software development: The case of Drupal and Twitter. *IEEE Software*, *31*(4), 72-80.

Watson, H. J. (2018). Revisiting Ralph Sprague's framework for developing decision support systems. *Communications of the Association for Information Systems*, *42*, 363-385.

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

Wei, K., Crowston, K., Li, N., & Heckman, R. (2014). Understanding group maintenance behavior in Free/libre open source software projects: The case of Fire and Gaim. *Information & Management*, *52*(3), 297-309.

Wen, W., Forman, C., & Graham, S. J. H. (2013). The impact of intellectual property rights enforcement on open source software project success. *Information Systems Research*, *24*(4), 1131-1146.

West, J., & O'Mahony, S. (2008). The role of participation architecture in growing sponsored open source communities. *Industry and Innovation*, *15*(2), 145-168.

Wittenbaum, G. M., Hollingshead, A. B., Paulus, P. B., Hirokawa, R. Y., Ancona, D. G., Peterson, R. S., Jehn, K. A., & Yoon, K. (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.

Xiao, X., Lindberg, A., Hansen, S., & Lyytinen, K. (2018). "Computing" requirements for open source software: A distributed cognitive approach. *Journal of the Association for Information Systems*, *19*(12), 1217-1252.

Appendix A: 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, due to their distributed leadership, FLOSS teams lack 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 someone proposed an alternative. Similarly, a certain person or institution generally did not need to authorize a decision. In the very few cases when a decision required such authorization (e.g., where the administrator or the project leader needed to handle a discussed issue), 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, an evaluation behavior, uniquely differs in asynchronous collaboration where team members test a proposed solution and post the results of their actions rather than simply posting opinions (Keen & Cummins, 1994). In a synchronous discussion, participants rarely have time to take such action during a meeting.

We present the final coding scheme for stages in the decision-making process in Table A1.

Table A1. Final Coding Scheme for Stages in Decision-making Process

Phase	Functional move	Explanation	Examples from the 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 & 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 & 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 & 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 & 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 & 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 & 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. See the example below.

Table A1. Final Coding Scheme for Stages in Decision-making Process

	Example for the emergent code: (E-2) Solution evaluation—action	A: Do you remember that bug I told you when you typed into a window and other person received that messages? …I think we will have to improve the multiple windows fix. I"ve been thinking of it [Then provides a potential solution D-3:] We should keep two variables for each window. One should be the list of connected users to that window, and another for the "last" user in that window….[Provides a code to solve the issue]
		
		B: [Provides "(E-2) Solution evaluation-action" by showing that they have physically tested the code provided by the Person A]:
		I've been checking the code, for the moment I've found a small error here, at the end of ccmsn_destroyed_msgwin:
		if { [info exists msg_windows([string tolower ${email})]] } {
		look at the order of the] and), it should be
		if { [info exists msg_windows([string tolower ${email}])] } {
		so that variable wasn't existing. I'm going to check a bit more, but do you think that could be a problem?	

	(E-3) Solution confirmation	This move describes the activity to ask for confirmation or initiate voting.	Solution confirmation (Poole & Roth, 1989)
(A) Announcement	(A-1) Decision announcement	This move announces the final decision on team level.	Decision product (Wood, 1986)

About the Authors

U. Yeliz Eseryel is an Assistant Professor of MIS at East Carolina University. She received her PhD in Information Science and Technology from Syracuse University (2010). Her research includes the theory and practice of open innovation enablement through information systems, Open Source Software Practices, and IT-Enabled Leadership. Her research has been published at *JSIS*, *JAIS*, *EJIS*, *EEE*, I&M, *JASIST,* and others. Her industry background includes business and strategy consulting and IT project management. She teaches internationally at the corporate, executive, masters, and undergraduate levels since 2004.

Kangning Wei is a Professor in the School of Management at Shandong University, China. Her research interests include social practices in Free/Libre Open Source Software development, user behavior in online communities and virtual collaboration in a broad way. She received her PhD in Information Science and Technology from Syracuse University, USA.

Kevin Crowston is a Distinguished Professor of Information Science in the School of Information Studies at Syracuse University. He received his PhD in 1991 in Information Technologies from the Sloan School of Management, Massachusetts Institute of Technology (MIT). His research examines new ways of organizing made possible by the extensive use of information and communications technology. His specific research topics include the development practices of Free/Libre Open Source Software teams and work practices and technology support for citizen science research projects, both with NSF support.