Communications of the Association for Information Systems

Research Paper 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
East Carolina University

Kangning Wei
Shandong University

Kevin Crowston
Syracuse University

**Abstract:**

Community-based FLOSS teams with internal governance are an extreme example of distributed teams, prominent in software development. At the core of distributed team success is team decision-making and execution. However, in the case of FLOSS teams, the lack of formal organizational structures to guide practices and the reliance on asynchronous communication might be expected to make decision making problematic. Despite these challenges many FLOSS teams are effective. There is a paucity of research in how organizations make IS development decisions in general, and the research in FLOSS decision- making models is particularly limited. Decision-making literature in FLOSS teams has focused on the distribution of decision-making power. Therefore, it is not clear which decision-making theories fit the FLOSS context best, or whether novel decision-making models are required. We adopted a process-based perspective to analyze decision-making in five community-based FLOSS teams. We identified five different decision-making processes, indicating FLOSS teams use multiple processes when making decisions. Decision-making behaviors were stable across projects despite different type of knowledge required. We help fill the literature gap about which FLOSS decision mechanisms can be explained 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 [editorial/peer] review. It was received xx/xx/20xx and was with the authors for XX months for XX revisions. [firstname lastname] served as Associate Editor. **or** The Associate Editor chose to remain anonymous.]

## 1. Introduction

This paper identifies decision-making processes in community-based Free/Libre Open Source Software (FLOSS) development teams with internal governance. As an essential component of team behavior (Guzzo & Salas, 1995), decision making has been extensively studied. Understanding decision-making processes in teams is important because the effectiveness of decision-making processes can have a large impact on overall team performance (Hackman, 1990). Decision making is of particular interest in information systems research because these processes are often supported and influenced by advanced information and communication technologies (ICT) (Huber, 1990; Shaikh & Karjaluoto, 2015).

Our study is set in the context of FLOSS teams. Researchers are interested in 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). FLOSS has been said to enable small businesses and users to play a role in democratized business software innovation within the business ecosystems (Allen, 2012). Andriole (2012) suggests that FLOSS is impactful because of the open architectures it encourages meaning that FLOSS 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 the challenges include the lack of deadlines for implementing feature or bug fixes, not being as well-established in some areas, and often imposing high barriers to entry 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 decision-making structures in the FLOSS teams are dynamic and consensus-driven (Nelson et al., 2006), we will take the recommendation by Nelson et al. (2006) to investigate the extent to which FLOSS decision mechanisms can be explained using classical theories from organizational structure or require new thinking.

However, FLOSS teams are not all the same: there are different types of FLOSS teams, which could affect the governance and decision-making structures, making it necessary to bound our study. West and O'Mahony (2008) identified two types of FLOSS communities, namely 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 identified by de Laat (2007), namely (1) spontaneous governance, (2) internal governance, and (3) governance towards outside parties. Spontaneous governance refers to small projects that are self-directing and that have no explicit or formal control or coordination mechanisms. Internal governance refers to projects that have existed for a period of time and have multiple participants, requiring coordination and control to achieve desired outcomes (Midha & Bhattacherjee, 2012). However, these projects are still governed from within the project team. The last type of governance is one where the projects are highly institutionalized, either because of a non-profit foundation to protect the project, or when a project works with companies, meaning that governance is determined by those parties. We focus our study on the second type of projects, those that have internal governance because they are big enough to have identifiable decision processes but these processes can still be emergent. We refer to this type of FLOSS teams as "community-based FLOSS teams with internal governance". In the remainder of this article, 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 was motivated by three distinct characteristics of the FLOSS context that we expected would pose barriers to effective decision-making, requiring novel decision-making processes.

First, community-based FLOSS teams with internal governance are generally virtual, as developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily by means of ICT (Crowston, Wei, Howison, & Wiggins, 2012). The extensive use of ICT changes the way members can interact and so how they make decisions (Kiesler & Sproull, 1992). A lack of shared context and numerous discontinuities in communication faced by virtual team members can hamper decision making (Watson-Manheim, Chudoba, & Crowston, 2002). While the 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.

In community-based FLOSS teams with internal governance, not only is the communication technologically mediated, but so too is the work itself. A further complication is that prior research has suggested that information technologies can affect the decision-making mechanisms (Kiesler & Sproull, 1992). Asynchronous communications make it impossible for the 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 sense-making and understanding become more difficult for the participants. On the other hand, we do not know to which extent asynchronous decision-making may allow for the use of novel decision-making processes.

Second, given the distributed nature of the work over different time-zones, decision-making processes must enable participants from all around the world to contribute despite time-zone differences. For instance, decision-making processes are usually asynchronous in nature. FLOSS teams in particular rely on information technologies, such as team discussion fora, websites, bug trackers and source code repositories, what Barcellini, Détienne and Burkhardt (2014) term discussion spaces, for communication, coordination and discussion of alternatives. The asynchronous collaboration provides an additional unexpected benefit. In FLOSS development teams, the knowledge base for the decision and the decision-making actions are widely accessible and those interested can contribute with their opinions and knowledge with minimal barriers. In comparison to traditional organizations, it has been found that more people in FLOSS development can share power and be involved in team's activities (Crowston et al., 2012). The more participants and more discussions are involved in the decision-making processes, the more knowledge is accessible and transparent to many others, which in turn, enables participants to work on their own, and contribute what they have done back to the FLOSS development teams.

Third, unlike organizational teams, prior literature has identified community-based FLOSS teams with internal governance as being self-organizing, autonomous and self-managed (West & O'Mahony, 2008), meaning that they are not managed with formal authority relations (O'Mahony & Ferraro, 2004a, 2007), i.e., their leaders are not externally appointed. Indications of ranks or roles are materialized through interaction rather than external cues, meaning that there is no hierarchical source of decision authority. In these teams, leadership is emergent and fluid 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 are said to eschew positional power. Eseryel & Eseryel (2013) found that the leaders 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). This makes decision-making in this setting even more important, as decisions are likely to contribute to leadership emergence, and provide the basis for organizing. Technical contributions that are valued by other team members (such as the number and popularity of the packages one maintains) can determine membership and leadership decision of those members. Yet, when communities impose rules such as face-to-face meetings, key signings, and recommendations by existing members for membership, such as in the case of the Debian community, these may influence the relative influence of technical contributions on membership and leadership, and introduce new factors such as the centrality within 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 of investigation. Further, they recommend the study of mechanics for artifact creation. In this article, following their advice, we are investigating the mechanics of decision making for software development. Examination of how decision-making processes are adapted in the face of these characteristics will extend our understanding of team decision making. Furthermore, understanding how the technological systems that support and constrain virtual work affect decision-making processes should be informative for many kinds of knowledge work, which becomes increasingly virtual. At a more specific level, knowledge of FLOSS decision-making process can be informative for organizations or firms collaborating with FLOSS teams (Santos, Kuk, Kon, & Pearson, 2013). FLOSS impacted the software industry significantly and many organizations develop and/or use FLOSS (Ven & Verelst, 2011). As organization's engagement in FLOSS development is not passive (Colombo, Piva, & Rossi-Lamastra, 2014), understanding the decision-making processes in FLOSS is critical for organizations hoping to extract the most values from their interactions with these communities. Our investigation on FLOSS decision making is in line with Feller and Fitzgerald's (2000) recommended research agenda which should focus on the development processes of FLOSS communities based on the traditional reporting questions such as who, what, where, when, why and how.

While decision making has been recognized as an important function in FLOSS teams (Crowston et al., 2012), prior literature has black-boxed FLOSS related decisions, and provided mainly the outcomes of the decisions rather than investigating the process of decision-making. To fill this gap, we explore how decision-making processes are structured in community-based FLOSS development teams. More specifically, based on the contingency model of decision-making processes (which will be discussed in detail in section 2), we answer the following research question:

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 within the extant FLOSS research. Then we introduce phase theories of team decision-making processes, which we use to analyze FLOSS data.

### 2.1 Overview of Extant FLOSS Research and Decision Making

To position our research within the current FLOSS literature, we provide a brief review of the FLOSS literature, with examples and we discuss how each stream addressed decision making. There is much research in open source software and numerous review articles 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). Our purpose is not to provide a comprehensive review of all the publications, but rather to give the reader a brief overview. We categorize FLOSS research in 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 area of FLOSS research as "FLOSS as an example of a unique phenomenon". Some examples include Love and Hirschheim (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, both individualist and collectivist perspectives on the locus of knowledge are needed. 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, meaning 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 type of FLOSS research focuses on company-level decisions. These decisions are 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 and colleagues theorized on OSS-based value-creation and value-capturing using inter-organizational networks (Morgan, Feller, & Finnegan, 2013; Morgan & Finnegan, 2014). Alspaugh, Scacchi, and Asuncion (2010) provided guidance for achieving best-of-breed component strategy while obtaining desired license rights when FLOSS software and proprietary software development efforts are combined. 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 OSS adoption decisions by SME's. Marsan, Pare, and Beaudry (2012) investigated the perceptions of IT specialists and their backgrounds that affect FLOSS adoption decision. Feller, Finnegan, and Nilsson (2011) found four typologies for FLOSS innovation process adoption by public organizations and relevant affect business models. Chengalur-Smith, Sidorova, and Daniel (2010) showed how infrastructure source openness influences FLOSS technology use decision, which in turn increases business value. Machado, Raghu, Sainam, and Sinha (2017) discussed how the existence of FLOSS alternatives affects the firms' pricing strategies and piracy control efforts. Similarly, August, Shin, and Tunca (2013) developed an economic model to jointly analyze the investment and pricing decisions of the originator companies of software and subsequent FLOSS contributors.

The fourth type of FLOSS research is conducted to investigate various processes at the project level, which is our focus. This stream of research focused on determinants of project success, project attractiveness, as well as the various processes used within projects such as innovation, knowledge creation, and requirements engineering. Much research at the project level 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 (2016) 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 their article 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 identifying 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 amount of work activities within projects. Xiao, Lindberg, Hansen, and Lyytinen (2018) investigated the requirements engineering process and showed how the distributed, dynamic and heterogeneous structure underlying 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 the 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 those 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, which is 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. A connection has been observed between hierarchical governance structure and centralization of decision-making processes (Gacek & Arief, 2004). The centralized decision-making process in Linux Kernel (Moon & Sproull, 2000) has been characterized as a benevolent dictatorship (Raymond, 1998). In contrast, the relatively non-hierarchical GNOME team has a decentralized decision-making process involving task-forces (German, 2003). Finally, roles and decision-making structures have been observed to be dynamic (Nelson et al., 2006; Raymond, 2001; Robles, 2004) and fluid (O'Mahony & Ferraro, 2004b). Fitzgerald (2006) suggested that early in the life of a team, a small subset will control decision making, but as the software grows, more developers will get involved.

As FLOSS developers tend to work on multiple projects, the fifth research stream focused on the influences on membership in multiple FLOSS projects on the project level outcomes. For example, Singh, Tan, and Mookerjee (2011) showed influences of project internal cohesion and external cohesion on project success as well as the influences of the project's external network's technological diversity. Peng and Dey (2013) found that co-membership among project teams is an effective mechanism for building network ties for knowledge sharing and further specified that leader-follower and follower-leader network ties are more beneficial to OSS success 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 type of FLOSS research includes FLOSS decisions at the individual level. The most common type of research within this category includes the participation 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 majority of the tasks are done by a single individual and those tasks that are too large for an individual get deferred. Ke and Zhang (2010) identified the influence of various types of motivations on the level of task effort put forth by the FLOSS developers. Choi, Chengalur-Smith, and Nevo (2015) investigated the influence of three community markers (ideology, loyalty and identification) on the behaviors of passive users such as user brand extension, word-of-mouth, community involvement and endorsement. Wen, Forman, and Graham (2013) showed how user interest and developer activity in FLOSS software are influenced by lawsuits. Singh, Tan, and Youn (2011) investigated how individuals with different learning states learn from their peers versus from their own learning experience. Factors that influence developers' code reuse decisions are investigated by Sojer and Henkel (2010), who 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 the decisions about adoption strategies, how to create value for companies, 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, in fact examine the general governance of FLOSS, such as who has decision-making power. However, there is a lack of empirical research that opens up the black box of FLOSS decision-making process. This gap is illustrated in the lack of coverage of the topic in published FLOSS review and research framework articles (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 decision-making process, it is important 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 users of the software as a whole to a future course of action, e.g., decisions about which bugs to fix and how or which features to add, as well as more strategic decisions related to social, organizational, strategic and legal aspects of development.

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

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

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

However, Poole and his colleagues suggested that the normative models are not adequate to capture the dynamic nature of decision-making sequences, and propose another class of phase models, multiple-sequence models (Poole, 1983; Poole & Roth, 1989a). 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 a study of 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, rather than being planned by the team ahead of time.

Multiple-sequence models of decision making are advantageous because they not only capture the complexity of the decision-making process that may vary due to factors such as task structure (Poole, 1983), but also provide a systematic approach to studying the dynamic decision-making processes (Mintzberg, Raisinghani, & Theoret, 1976; Poole & Roth, 1989a). 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. We therefore adopted this approach in this paper.

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

## 3 Research Method

We turn now to the design of 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 content analyzed the episodes 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. As previously noted, FLOSS business models are diverse. To control unwanted systematic variance, we chose community-based projects with internal governance structure that were roughly similar in age, and that were all at production/stable development status. Projects at this stage have relatively developed membership and sufficient team history to have established decision-making processes, yet the software code still has room for improvement, which enables us to observe rich team interaction processes around development. Acknowledging that the development tools used might structure the decision-making processes, we selected projects that were all hosted on SourceForge (www.sourceforge.net), a FLOSS development site popular at the time of data collection that provides a consistent ICT infrastructure to developers. Table 1 below provides the overview of selected cases, which are described further below.

Therefore, we picked two different types of software, where the participants' knowledge about the software differs, which may in turn influence the patterns of interaction and decision-making. Specifically, we selected projects that developed Instant Messenger (IM) clients and Enterprise Resource Planning (ERP) systems, expecting that these two types of projects would be different in complexity, which in turn would affect the decision-making processes.

We initially chose 3 cases for each project type: Gaim (currently known as Pidgin), aMSN and Fire from IM projects, and Compiere, WebERP and OFBiz (currently known as Apache OFBiz[1]) from ERP projects. However, during data analysis we came to realize that Compiere was not a community-based project like the others, since it was started by a company and now has both community and commercial aspects in its development. Therefore, it would be better classified as a team with governance toward outside parties based on the governance categorizations of de Laat (2007). To avoid possible bias introduced by this project, we decided to remove this project from our study, resulting in 5 (3 IM and 2 ERP) projects in the final design.

**Table 1. Project Comparison**

| **Project Name / Category** | **Gaim[2] (Pidgin)** | **Fire** | **aMSN** | **WebERP[1]** | **OFBiz[3]** |
|---|---|---|---|---|---|
| **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[4] | 27 | 35 |
| **Initial Release** | November 1998 | April 1999 | May 2002 | January 2003 | November 2001 |

ERP systems are some of the most complex software (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 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

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

[2] Most of the data on Gaim (Pidgin), OfBiz and WebERP were collected from Openhub.net using the compare projects function.

[3] Source: https://www.openhub.net/p/Apache-OFBiz

[4] Source: http://www.amsn-project.net/current-developers.php

email module and security module. OFBiz provides the features of product and catalog management, promotion and pricing management, supply chain fulfillment, contracts, payments and billing, which are functions that are spread between sales, marketing, customer management, supply chain, accounting and finance. Each of these areas requires unique domain knowledge, in addition to technical knowledge. Rettig (2007) suggested that these systems are so complex that developing and changing them becomes risky because no single person within 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 the software can be engineered to fit the needs of diverse companies. Second, ERP systems integrate high volume of data, which were earlier either unavailable or impossible to derive with other software (Chaudhari & Ghone, 2015). Further, the level of automation that ERP provides (Haddara, 2018) adds to the complexity of the software. Glass (2003, p. 58) suggests that for every 25% increase in complexity in the tasks to be automated, the complexity of the software rises by 100%. As a result of these factors, ERP systems are "massive programs, with millions of lines of code, thousands of installation options and countless interrelated pieces, [and thus they] introduced new levels of complexity" (Rettig, 2007, p. 23). Part of the complexity comes from the sheer size of these programs as indicated by the lines of code included (1.5M and 6.5M for OFBiz and WebERP). A Carnegie Mellon Study finds that the average professional coder makes 100 to 150 errors for every 1,000 lines of code, (Mann, 2002). That means for an ERP system such as WebERP of 6,5 million lines of codes, there could be anywhere between 650,000 to 975,000 bugs to fix as the software is being developed.

In contrast, IM clients have one main function and a handful features. The knowledge that the developers need may be purely experiential based on their own use, in serving the needs of many. Their code base may be in the thousands of lines of code compared to millions of lines for ERP systems. Many more individuals may participate in the programming, due to lower levels of skills needed, and therefore lower barriers to entry. Therefore, it is expected that the IM projects have relatively simpler decision processes, where fewer individuals' inputs are needed, for example. To sum up, we expect that due to the differences in the types and variety of knowledge needed between ERP and IM software, we expect the decision-making processes in ERP projects to differ from those in IM projects.

## 4 Identification of the Patterns of Decision-Making Process

In the following sections, we describe the research method in each phase and report the findings in detail. Specifically, section 4 describes the qualitative design 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, and where our findings may be more easily applied to and generalized in other similar yet non-FLOSS contexts. Our goal was to identify generic decision-making processes for strategic and tactical decisions, which may then be tested at other communication and decision-making tools used by FLOSS teams. In collecting strategic versus tactical decision episodes we used the following definitions: We defined tactical decisions as decisions where the central issues are related to an indication for a change in the software code. This included an acceptance of a patch or lines of code that will become part of the code base. We defined the strategic decisions as the decisions where central issues are not code related. The topics of decisions include legal issues, membership issues, funding, maintaining a positive group atmosphere, and software architecture.

Before we collected our data for this study, we followed a number of venues for decision-making, including issue trackers, instant messaging tools of projects such as GAIM and the developers' fora[5]. We observed that developers' fora were best in their coverage of both strategic decision and tactical decisions, whereas the use of issue trackers were solely limited to technical issues such as solving bugs or new features, and

[5] During our data collection, none of the projects were using GitHub.

the instant messaging tools such as Internet Relay Chat (IRC) were typically[6] used to ask for advice on an area that a developer is stuck on, rather than for making decisions at the group level. This decision of collecting data from a tool that hosts both strategic and tactical decision-making is in line 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: Key studies that informed this literature stream were conducted with strategic decision-making teams as well as with student teams making tactical decisions (e.g., Poole, 1983).

Data were obtained from the SourceForge website. Our analysis of the developers' fora interactions regarding the decisions we analyzed did not reveal references to off-line discussions, suggesting that this data source provided a complete view of the decision-making process, at least for the decisions analyzed for this study. Furthermore, we intently 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 were able to observe full decision-episodes in the developers' fora.

As our primary unit of coding and analysis, we selected the decision episode, defined as a sequence of messages that begins with a decision trigger that presents an opportunity or a problem that needs to be decided and that includes the required acts of issue discussion and which possibly ends with a decision announcement (Annabi, Crowston, & Heckman, 2008). To give an example, a decision trigger may be a feature request or a report of a software bug. A decision announcement may be either a statement of the intention to do something or an actual implementation of a fix. Note that some decision processes did not result in a decision that was announced to the group, while others had multiple announcements as the decision was revised. The messages in an episode capture the interactions among team members that constitute the process of making a particular decision from start to finish.

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

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

[6] It was our observation that IRC was generally used to get quick help from fellow coders. However, we would like to acknowledge that one anonymous reviewer had noticed some decisions being made on IRC in their research. For this reason, it is very important for researchers to be familiar with the practices used by the community they are researching.

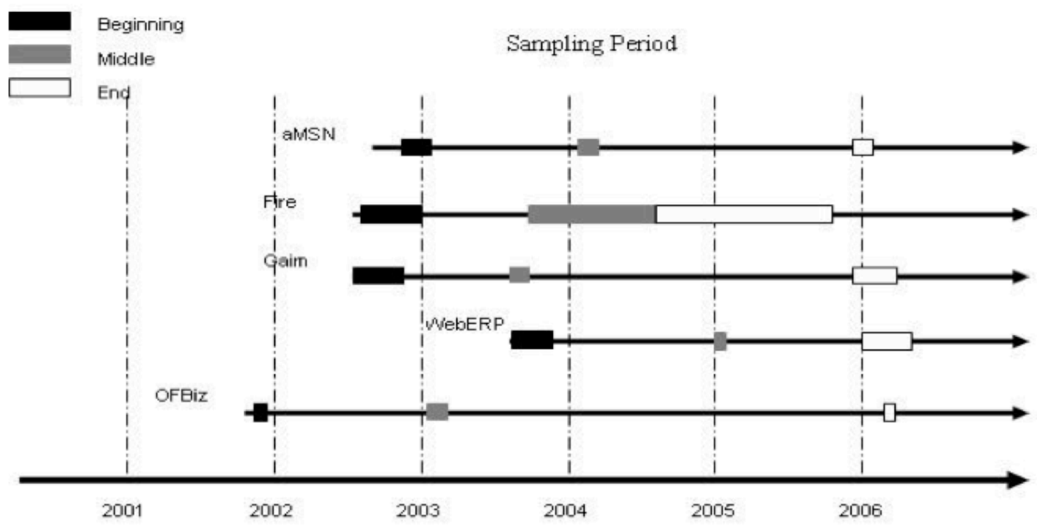

**Figure 1. Sampling Periods for Decision Episodes by Project**

In investigating decision-making processes, it is important to take into consideration that the dynamics of decision-making in community-based FLOSS projects develop over time due to the nature of participation among voluntary community members. Benbya and Belbaly (2010) show that both the type of participation and the level of effort by the individuals differ based on their motivation to gain knowledge on a specific area. Further, the type of individual's participation to the decision-making process may change based on how much knowledge they have in an area. Accordingly, sampling of decision episodes was stratified by time: we chose 20 episodes from the beginning, middle and end periods of each project[7] based on a concern that the decision-making process might be different at different stages of the software development (e.g., initial collaboration vs. a more established team). However, $\chi^2$ 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.

The result of this initial coding process was a collection of 300 decision episodes, each including a number of messages with a trigger and (when present), one or more decision announcement(s). The sample size was chosen to balance analysis feasibility with sufficient power for comparisons. With 60 episodes per project, we have reasonable power for comparison across projects while keeping the coding effort feasible.

## 4.2 Coding Scheme Development for Decision Processes

Once we had a sample of decision episodes, we content analyzed them by coding the segments of text that embodied the decision-making steps to identify decision-making process in each episode. The coding scheme was developed deductively in two steps. First as noted above, we adopted the Decision Functions Coding System (DFCS) developed by (Poole & Roth, 1989b). This coding system uses as the primary unit of coding the "functional move", which is defined as "the function or purpose served by a particular segment of the conversational discourse" (Wittenbaum et al., 2004). Functional moves have been used extensively to understand the nature of interaction in both face-to-face and computer-mediated environments (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

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

threaded discussion fora. We used functional moves to identify the function of messages in each episode. Note that a single message might include zero, one or multiple functional moves.

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

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

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

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

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

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

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

Figure 2 provides an example of how these functional moves were coded based on an example from the Gaim project. This process went through all four phases of identification, development, evaluation and announcement consecutively, however making loops back twice from the evaluation stage to the previous development phase. While many dynamic decisions loop back almost at every stage, for simplicity, we chose to show an example where only two loop-backs happened.

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

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

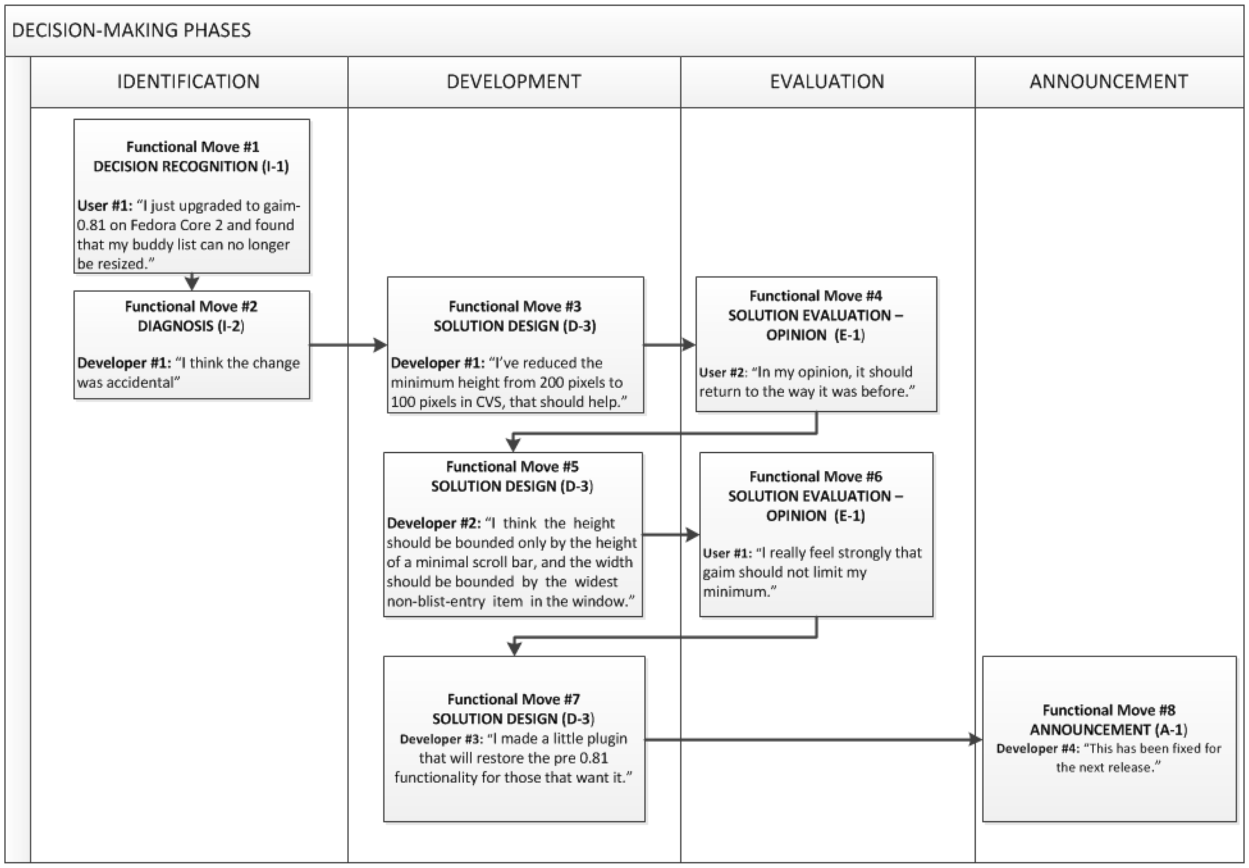
DECISION-MAKING PHASES
IDENTIFICATION
DEVELOPMENT
EVALUATION
ANNOUNCEMENT
Functional Move #1
DECISION RECOGNITION (I-1)
User #1: "I just upgraded to gaim-0.81 on Fedora Core 2 and found that my buddy list can no longer be resized."
Functional Move #2
DIAGNOSIS (I-2)
Developer #1: "I think the change was accidental"
Functional Move #3
SOLUTION DESIGN (D-3)
Developer #1: "I've reduced the minimum height from 200 pixels to 100 pixels in CVS, that should help."
Functional Move #4
SOLUTION EVALUATION – OPINION (E-1)
User #2: "In my opinion, it should return to the way it was before."
Functional Move #5
SOLUTION DESIGN (D-3)
Developer #2: "I think the height should be bounded only by the height of a minimal scroll bar, and the width should be bounded by the widest non-blist-entry item in the window."
Functional Move #6
SOLUTION EVALUATION – OPINION (E-1)
User #1: "I really feel strongly that gaim should not limit my minimum."
Functional Move #7
SOLUTION DESIGN (D-3)
Developer #3: "I made a little plugin that will restore the pre 0.81 functionality for those that want it."
Functional Move #8
ANNOUNCEMENT (A-1)
Developer #4: "This has been fixed for the next release."

Figure 2 An Example Illustrating How a Decision Episode is Coded for Functional Moves

## 4.3 Findings: Qualitative Analysis of Decision-Making Patterns

Following the procedure described in section 4.2, we sorted the 300 decision-making episodes into 5 clusters according to the number of phases. We labeled these processes as short-cut, implicit-development (implicit-D), implicit-evaluation (implicit-E), complete, and abandoned decision processes (i.e., lacking a final decision announcement). Figure 3 depicts the patterns of the five processes. The dashed lines in the figures indicate points at which there might be loops, leading to iterative decision process. The loop from decision announcement to previous phases indicates that one or more intermediate decisions were announced before the decision was finalized.

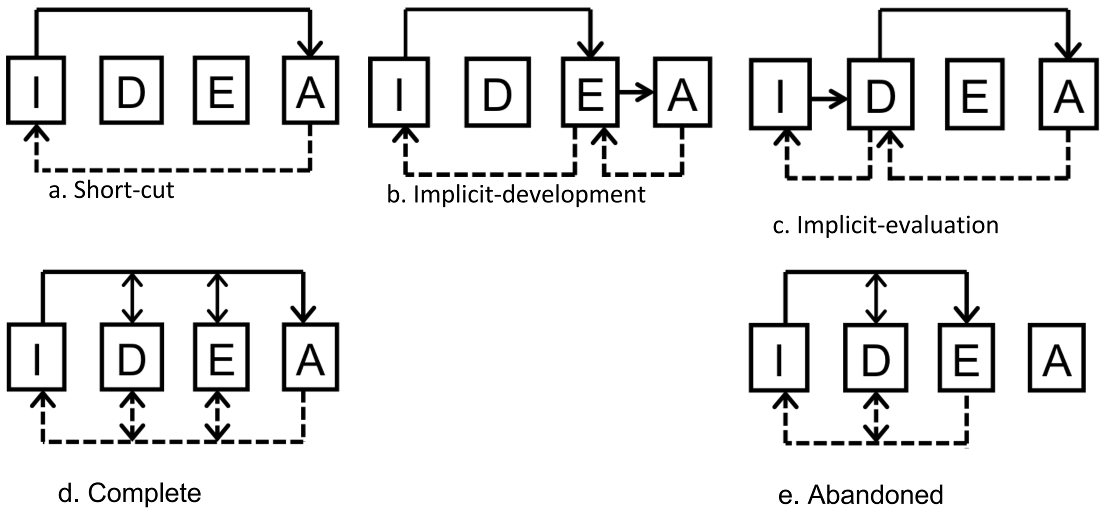

Figure 3. Five Decision-Making Processes Identified based on the Data

**Short-Cut** (Figure 3a). This process represents the simplest pattern, in which a decision is made right after opportunity recognition and perhaps a brief problem diagnosis, with no explicit solution development or evaluation. Examples of this kind are often observed in the bug report or problem-solving discussions in software-modification decisions. For example, in one decision episode in the WebERP project, a user reported a bug (code I-1, Decision Recognition), which was quickly followed by the response of an administrator that he "just fixed it" (A-1, Decision Announcement), with no further discussion or evaluation. While there is an absence of team input, we argue that these decisions are still team decisions, for two reasons: 1) Since all team members can view the bug fix and reverse it if they see it as inappropriate, a lack of reversal indicates an implicit consensus on the proposed course of action; and 2) the decision (e.g., a bug fix) affects the shared team output and binds the team to a future course of development (i.e., there are team consequences).

**Implicit-D** (Figure 3b). In this process, the solution development phase is skipped, which does not mean that a solution was not developed, but rather that there is a lack of evidence of the development phase in the online discussions. For example, in these episodes, the person who brings up an issue may have already done a diagnosis and provides a solution together with the issue. The subsequent discussions concentrate on evaluating the feasibility or the benefits and disadvantages of the suggested implementation, rather than looking for more alternative solutions. For example, in the aMSN project, a user wrote 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 steps of solution analysis, search and design were not visible in the text. However, these steps were conducted at least by the user who sent the patch, and possibly by others who did not feel it was necessary to report their progress.

**Implicit-E** (Figure 3c). The third type of decision-making process is called “Implicit-Evaluation”, indicating a lack of online evidence of evaluative discussion. In these episodes, a decision is announced directly after the solution alternatives are generated without explicit evaluation of the alternatives. For example, in aMSN, an administrator brought up a technical issue (I-1, Decision recognition) and proposed three solutions (D-3, Solution design). Most of the subsequent messages concentrated on determining whether the problem was one for the aMSN project or just a problem from its supporting software such as a KDE problem (I-2, Diagnosis). After some discussion and testing, members confirmed it was an aMSN tray icon problem (I-2, Diagnosis). The team attention then returned to suggesting alternative solutions (D-3, Solution design) and the problem was quickly fixed (A-1, Decision announcement).

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

Iterative processes were observed when the issue was more complex. The complexity of the issue stems from the fact that its diagnosis and resolution are tied to other sub-issues. As the sub-issues are interrelated, discussions may loop back to any previous phase at any time. It might sometimes be possible to find another trigger that could be interpreted as starting a new decision episode within these issues. However, since the issues are interrelated, it would not be faithful to the original source of the issue to treat them as different episodes. For example, in OFBiz project, one administrator started a thread about how to design a workflow and based on which specifications. His first question was “The first was, which activity should we start with, and how do we know when we're done?” (I-1, Decision recognition). He then went on to show that he looked for existing solutions: “I did find examples of workflows at WfMC including mail room, order processing, and various other things. It appears that the first activity for a given process is the first in the list.” (D-2, Solution search). He then described how the solution would apply to this setting and evaluated this option, indicating it may be an easy-to-change temporary solution by saying “At run time it will already be there so if another spec does it differently, or we find another way (the correct way?), it will be easier to change.” (E-1, Solution evaluation-option). Another administrator took the process back to the development stage by writing an example of how the start activities might work (D-3, Solution design) and then evaluated this option. The first administrator then said “What you said about starting and ending makes a lot of sense. That's a good idea of specifying a default start activity, and for each activity specifying whether or not it can be a start activity.” (E-1, Solution evaluation, opinion) and announced the solution (A-1, Decision announcement). However, then a user jumped in to recommend an alternate solution, taking the team from decision announcement back to the solution development stage. When the administrator mentioned the user’s solution would not work, the user improved his solution, leading to several loops of development and evaluation before a solution was agreed on.

**Abandoned** (Figure 3e). We called the final category “Abandoned decision-making process”. In these processes, no decision was announced by the end of the observed decision episode. Abandonments may occur in any phase of discussion and happen for various reasons. A decision-making process may be abandoned during the identification phase due to a disagreement on whether there is a real problem or if there is a need to fix it. It may be abandoned during the development phase due to disagreement about the merits of different technical approaches and concerns. Abandonment in the evaluation phase can be due to multiple parties pursuing individual interests. For example, in the Gaim project, an administrator suggested adding audio functionality to the product (I-1, Decision recognition). Several core members challenged the availability of this functionality (I-2, Diagnosis). The discussions revealed two different preferred solutions—releasing a stable version with minor changes or releasing an unstable version with a major innovation (D-1, Solution analysis). Both sides extensively examined the current solutions, took relevant consequences into account and provided feasible suggestions (D-3, Solution design, E-1, Solution evaluation-opinion). However, after 11 days of discussion, we found no final decision announcement (even searching the list for months after).

Table 2 shows the distribution of the five decision-making processes across the 300 decision episodes. From the table we can see that, only 38% of decisions episodes analyzed went through all four phases (labeled as “Complete”), while 52% of the discussions reached a decision while skipping one or two phases

(Short-cut, Implicit-D or Implicit-E). No decision was reached in the remaining 10% of cases (Abandoned). In 23% of the decision episodes, the team decided right after the decision trigger was recognized (short-cut process). While 28% of decisions were made without the evaluation phase (Implicit-E process), only 1% of the decisions were made without a visible development phase (Implicit-D process).

**Table 2. Count of Observed Decision Processes for All Episodes**

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

When we looked for differences in the patterns exhibited by the ERP versus IM projects, we have not observed any systematic difference in the decision processes. $\chi^2$ tests[8] (Table 3 and Table 4) showed similar patterns in the use of different decision processes between IM and ERP projects for both tactical decisions ($\chi^2$ =1.644, p = 0.649) and strategic decisions ($\chi^2$ = 6.521, p = 0.100). The different types of knowledge required by the ERP FLOSS developers did not seem to cause them to use different phases or functional moves than those explained above and provided in the coding scheme in Appendix 1. Therefore, we concluded that the type and extent of knowledge required for the software does not influence the decision processes used by the FLOSS development team.

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

| | Short-Cut | Implicit-E | 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%) |
| | 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**

| | Short-Cut | Implicit-E | 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%) |
| | 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% of decisions followed a linear decision process, while the other 61% included one or more loop backs, following an iterative decision process.

## 5 Discussion of Findings & Theoretical Contributions: Multiple Sequences of Decision-Making Processes in FLOSS Development

In this study, we investigated decision-making process due to the focus of information systems research on how decision processes are influenced and supported by information technologies (Huber, 1990; Shaikh & Karjaluoto, 2015). This is true especially for community-based FLOSS projects with internal governance, where decisions are made virtually, asynchronously, across different time-zones and depending almost exclusively on information systems (Crowston et al., 2012). The best way of enabling and supporting the virtual, asynchronous decision-making that spans different geographical locations and time-zones requires an in-depth understanding of what the decision-making processes are. Only then, the right types of new

[8] Since there were only 4 cases of implicit-D episodes, they were excluded from this analysis.

information technologies can be identified that supports the decision processes at hand. Group Support Systems is a highly funded (by both grants and the industry) subset of Decision Support Systems Research, which focuses exactly on the area of developing information systems that support the decision-making process within groups (Arnott, Pervan, & Dodson, 2005). Watson (2018) reminded us 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 the phases of intelligence, design and choice, we found in FLOSS teams four phases, which incorporates the distinct subsets of development stage (solution development) and evaluation stage (evaluation of the developed solutions). Further, we identified noticeable sequences in the decision-making process such as skipping of phases, and iteration back to earlier phases as described in more detail below.

Our key contribution to decision-making literature is the identification of the five different decision-making processes observed in community-based FLOSS development teams with internal governance. Identification of these processes helps fill in the gap in the literature identified by Nelson et al. (2006) on the lack of investigations regarding the extent to which FLOSS decision mechanisms can be explained using classical theories from organizational structure. The extant research on FLOSS 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 including FLOSS (Deodhar et al., 2012), company level decisions such as on 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), pricing decisions (August et al., 2013; Machado et al., 2017). However, none of these decisions at the company level opened the black-box of these processes and explicated the decision process sufficiently to provide an input for the facilitation of these processes. Similarly, the decisions that were 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), FLOSS leadership process (Eseryel & Eseryel, 2013) do not explicate the decisions sufficiently to identify elements of decision-making, that can then be supported with group decision making technologies. Lastly, the individual level FLOSS research focuses more on the elements that support individual contribution decision, rather than investigating the decision-making process itself. The investigated factors that contribute to the individual participation decision 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), lawsuits related to the FLOSS (Wen et al., 2013), to name a few. This stream of research does not investigate what decision processes individuals go through after they make the decision to participate in FLOSS, which is what 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 with which these processes were used. We saw decision-making processes in community-based FLOSS development with internal governance as having multiple sequences that reflect the unique characteristics of FLOSS setting. In this research, we identified 5 different decision-making processes varying in both numbers and sequences of decision-making phases: short-cut, implicit-development, implicit-evaluation, complete and abandoned processes. Four patterns were observed in the use of these processes: frequent short-cuts, frequent implicit-evaluation, infrequent implicit-development and many cycles looping back to previous stages in decision-making. We explain these patterns of different decision-making processes based on 1) the unique characteristics of FLOSS development and 2) the high level of dependency of FLOSS decision process on information technologies.

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

At first, these results seem to be a paradox: open projects that make decisions in a seemingly opaque and non-participatory fashion. Our finding of high level of reliance on short-cut processes, which is an individual decision-making process, as a highly common way of making decisions that are binding to 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 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 detailed evaluation of potential alternatives in advance. This value is reflected in a description of the Internet Engineering Task Force decision process (part of the cultural heritage of FLOSS): "We reject: kings, presidents and voting. We believe in: rough consensus and running code" (Clark, 1992, p. 543). The result is a set of decision processes that emphasize making a sufficiently good decision based on as much collaboration as needed rather than spending too much time for evaluating options to find a perfect solution through 100% contribution by everyone to the decision.

Secondly, the missing phases may also be an empirical support for the stigmergic coordination in FLOSS development (Bolici, Howison, & Crowston, 2015). By examining those decision episodes using simpler decision processes, we found that many of them had mentioned or referred to software codes explicitly in their discussion. Prior research had proposed that stigmergic coordination makes explicit discussion unnecessary (Crowston, Østerlund, Howison, & Bolici, 2011; Robles, Merelo, & Gonzalez-Barahona, 2005). Namely, the shared and transparent nature of the information artifact and the technical ability to reverse code-submissions in case of disagreements enable the reliance on short-cut decision-making processes. The short-cut decision-making process is 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 the idea of stigmergic coordination has been discussed in prior literature (Crowston et al., 2011; Robles et al., 2005), no empirical research has been conducted examining this question. With shared work products and discussion based on asynchronous communication, developers can work independently to determine and test solutions rather than needing to immediately discuss them with others, a decoupling that enables distributed voluntary contributors to be effective participants.

Moreover, we found that developers often raised questions about others' actions based on their knowledge, leading back to previous phases of decision-making, resulting in a high proportion of cyclic processes (181 out of 300, 61%). While our findings are in line with observation that "IS decisions are often complex and dynamic" (Boonstra, 2003, p.206), the factors that are previously used to explain this cyclicity, such as political influences, urgency and necessity (Eisenhardt & Zbaracki, 1992a; Eisenhardt & Zbaracki, 1992b; 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, meaning that anyone can observe and contribute to a decision in process, even joining later a discussion that has been going on for some time. This pattern may also reflect the fact that no individual organizes the discussions to follow a normative path, as would be observed in teams with managers or decision support systems to structure the decision process.

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

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

Identification of these processes is important because the decision process used by the group directly affects group performance. Such an in-depth examination of the microstructures of decision-making processes compliments existing macro-level research on decision-making (e.g. German, 2003; Raymond, 1998). The frequency and type of decision-making processes used by FLOSS teams can be inputs for future theory development efforts predicting group performance. For example, quantitative studies can compare the types of decision processes used and the decision effectiveness (or overall project success).

Lastly, based on the earlier 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 had influenced our case selection strategy. However, contrary to our expectation, we did not observe differences in the decision-making processes used between simple (IM) and complex (ERP) software projects. Thus, our findings suggest that FLOSS projects tend to adopt similar decision-making processes for decisions regardless of the complexity and the knowledge requirements of the software developed by the FLOSS communities. This similarity reflects the observation that the software development process seems to be organized similarly across projects: using same sets of ICT tools in discussion and implementation spaces, parallel development and debugging which involve loosely-centralized and gratis contribution from individual voluntary developers (Feller & Fitzgerald, 2000), resulting in developers selecting similar scope of problems to work on, with similar decision demands. This finding suggests that the decision processes identified can be generalized 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 there are various types of FLOSS teams that are governed differently. Our study specifically focused on FLOSS teams that have internal governance, which we called "community-based FLOSS teams". These projects have existed for a period of time, they have multiple participants, and their governance is from within the project team and they require coordination and control to achieve desired outcome. Therefore, our findings should be tested for FLOSS teams that may have different types of governance to see if they can be extended to these teams. Specifically, FLOSS teams that are relatively new, and those teams that are highly institutionalized either because of a non-profit foundation or those that are formed by companies may show different decision-making dynamics. Therefore, the decision-making processes we identified should be tested in these two types of FLOSS teams for generalizability for those settings.

Secondly, we limited our investigation of decision-making processes to the discussion fora. The advantage of this approach was that it helped us capture both strategic and tactical decision-making processes. Different communication media may provide different affordances (Volkoff & Strong, 2013). Therefore, future research should test the five decision-making processes that we have identified in different types of communication media, such as issue trackers or pull requests that are used by the FLOSS teams. The numerous communication media that FLOSS teams use include setting up various automated listservs that automatically send emails whenever a new patch is committed, specialized listservs for individuals working on translations, updating team website or team wiki, to name a few. Further, issue trackers help coordinate decisions on technical issues such as a bug report or an enhancement request. Some FLOSS teams use GitHub, which enables pull requests to create various changes on a branch. Pull requests may be used to discuss, review and edit various changes that are done on a commit, before these changes are finalized and committed 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 mentioned above have different features, each tool may provide different affordances, meaning different possibilities for action may be offered to users by different communication media (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 the issue trackers, focus only on technical decisions, and have a unique structure that forces the users to fill in different fields, and therefore may affect the organization of the decision-making process. GitHub tools may be more relevant to those people who focus on a subset of the project, such as the website development, or the development of a specific branch. We expect to see similar decision-making processes across different media because what we investigated were social practices supported by the information and communication technologies. 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 the decision-making processes we identified should be tested across different media.

A limitation of this study is the exclusion of synchronous discussion fora, such as IRC, Instant Messaging or phone calls. We have followed IRC and instant messaging channels of especially the IM projects such as Gaim before we made the decision to focus on the developers' fora. Our observations of these channels informed us that these channels were typically used to clarify programming questions quickly, rather than making decisions. This observation is the reason why we decided to focus on developers' fora for investigating community-level decision making.

We had no way of observing one-to-one IM conversations that happen outside of the publicly shared ones. Thus, we want to acknowledge that it is possible that some of the steps in the decision-process that we infrequently observed were in fact carried out by a subgroup using such alternative channels. Future research should consider the impact of communications synchronous communication channels on community-level decision making on developers' fora. However, we would argue that the use of such channels would not change our main conclusion, namely that many decisions that bind the teams to a course of action are made without explicit involvement of the entire team in seemingly important phases of the decision process.

Another limitation of this research is the small sample size (i.e., five projects and 300 decision episodes). While it enabled us to conduct manual coding and provided us with rich data that increased our understanding of the decision-making process from different projects, it limited the types of statistical analysis we could run with our data. For example, we only used two types of FLOSS projects (i.e., IM projects vs. ERP projects) thus limiting the generalizability of the result. We specifically focused on these two projects because they represented two extremes on the continuum of variety of knowledge required for decision-making: While the 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 users of the software.

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

## 6 Practical Implications

Three groups of individuals in the practitioner community can benefit from the results of this study: 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) those who would like to bring to their organizations the FLOSS model of work, where internally governed small communities, such as those investigated in this article, collaborate on technical projects.

Understanding decision-making processes also enables the creation of group decision support systems and other information systems that would fulfill the team requirements. For instance, if FLOSS team members use applications such as the Algorithmic Autoregulation software (Fabbri et al., 2014) and habitually record their coding processes as they do their work, this would help explicate the individual decision-processes that make up the short-cut decisions. Discussion and implementation spaces are especially crucial to the success and continuity of distributed teams such as FLOSS, which depend on such systems for both task accomplishment and group maintenance.

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

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

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

## 7 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*(28), 483-496.

Almarzouq, M. (2005). Open source: Concepts, benefits, and challenges. *Communications of the Association for Information Systems, 16*(37), 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.

Andriole, S. J. (2012). Seven indisputable technology trends that will define 2015. *Communications of the Association for Information Systems, 30*(1), 61-72.

Annabi, H., Crowston, K., & Heckman, R. (2008, 14–17 December). *Depicting what really matters: Using episodes to study latent phenomenon.* Paper presented at the International Conference on Information Systems (ICIS), Paris, France.

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*(1), 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*(30), 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.

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

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

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. *Journa 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*. Paper presented at the Twenty-Fourth Internet Engineering Task Force, Cambridge, MA.

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. doi:10.1016/j.respol.2013.08.015

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

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*, 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), 1069-109**6**.

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).

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. (1992a). Strategic decision making. *Strategic Management Journal, 13*(52), 17-37.

Eisenhardt, K. M., & Zbaracki, M. J. (1992b). Strategic decision making. *Strategic Management Journal, 13*, 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., Zambianchi, L., . . . Thumé, G. S. (2014). The algorithmic autoregulation software development methodology. *Revista Eletrônica de Sistemas de Informação, 12*(3), paper 2.

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

Feller, J., & Fitzgerald, B. (2000). *A framework analysis of the open source software development paradigm.* Paper presented at the Proceedings of the twenty first international conference on Information systems, Brisbane, Australia.

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

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: Pearson Education.

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: 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, California: 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*, 96-123.

Love, J., & Hirschheim, R. (2017). Crowdsourcing of information systems research. *European Journal of Information Systems, 26*, 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*, 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*, 756-783.

Mann, C. (2002). Why software is so bad. *Technology Review*(July-August), 32-38.

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*, 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. *Adminstrative 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*, 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*, 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*(12), 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 AIS, 2006*(18), Article 7.

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*(8), 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*. Harvard Business School & IESE Business School. Retrieved from

O'Mahony, S., & Ferraro, F. (2007). The emergence of governance in an open source community. *The Academy of Management Journal, 50*(5), 1079-1106. doi:10.5465/AMJ.2007.27169153

Parr, A. N., Shanks, G., & Darke, P. (1999). Identi�cation 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: 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). Thousands 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. (1989a). Decision Development in Small Group IV: A typology of Group Decision Paths. *Human Communication Research, 15*(3), 323-356.

Poole, M. S., & Roth, J. (1989b). Decision development in small groups 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*, 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. *Open Source Yearbook*.

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

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), 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*(9), 117-140.

Volkoff, O., & Strong, D. M. (2013). Critical realism and affordances: theorizing IT-associated organizational change processes. *MIS Quarterly*(37), 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*, 119–129.

von Krogh, G., Haefliger, S., Spaeth, S., & Wallin, M. W. (2012). Carronts 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*.

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

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

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

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

| Phase | Functional Move | Explanation | Examples from Literature |
|---|---|---|---|
| (I) Identification | (I-1) Decision recognition routine | This move recognizes an opportunity that may lead to a decision.<br>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, 1989a); 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, 1989a); diagnosis (Mintzberg et al., 1976) |

| Phase | Functional Move | Explanation | Examples from Literature |
|---|---|---|---|
| (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, 1989a) |
| | (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, 1989a) |
| | (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, 1989a) |
| (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, 1989a) |
| | (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.] |
| | 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]<br><br>B: [Provides "(E-2) Solution evaluation-action" by showing that they have physically tested the code provided by the Person A]:<br>I"ve been checking the code, for the moment I"ve found a small error here, at<br>the end of ccmsn_destroyed_msgwin:<br>   if { [info exists msg_windows([string tolower ${email})]] } {<br>look at the order of the ] and ), it should be<br>   if { [info exists msg_windows([string tolower ${email}])] } {<br>so that variable wasn"t existing. I"m going to check a bit more, but do you<br>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, 1989a) |
| (A) Announcement | (A-1) Decision announcement | This move announces the final decision on team level. | decision product (Wood, 1986) |