Free/Libre Open Source Software Development: What We Know and What We Do Not Know

KEVIN CROWSTON, KANGNING WEI,
JAMES HOWISON & ANDREA WIGGINS
Syracuse University School of Information Studies

We review the empirical research on Free/Libre and Open Source Software (FLOSS) development and assess the state of the literature. We develop a framework for organizing the literature based on the input-mediator-output-input (IMOI) model from the small groups literature. We present a quantitative summary of articles selected for the review and then discuss findings of this literature categorized into issues pertaining to inputs (e.g., member characteristics, technology use and project characteristics), processes (software development and social processes), emergent states (e.g., trust and task related states) and outputs (e.g. team performance, FLOSS implementation and project evolution). Based on this review, we suggest topics for future research, as well as identifying methodological and theoretical issues for future inquiry in this area, including issues relating to sampling and the need for more longitudinal studies.

Categories and Subject Descriptors: D.2.9 [**Software Engineering**]: Management – *Life cycle*; D.2.9 [**Software Engineering**]: Management – *Programming teams*; J.4 [**Computer Applications**]: Social and Behavioral Sciences - *Sociology*; K.6.3 [**Computing Milieux**]: Management of Computing and Information Systems – *Software management*
General Terms: Design, Human Factors, Management, Measurement, Performance
Additional Key Words and Phrases: Free/Libre open source software, development, computer-mediated communication, distributed work

1 INTRODUCTION

In this paper, we review the published empirical research literature on development of Free/libre and Open Source Software (FLOSS) development. FLOSS is an umbrella term that covers a diversity of kinds of software and approaches to development, so we start this review by clarifying our focus. Strictly speaking, the term free software[1] [1] or open source software [2] refers simply to software released under a license that permits the inspection, use, modification and redistribution of the software's source code [1, 2], and in the case of free software, that guarantees that these rights apply also to derivative works [1]. While there are important difference between these two kinds of software and their development communities [3], our focus is in this paper is on research on their development processes, which are largely similar [1], hence our use of this umbrella term.

[1] Sometimes referred to as "libre software" to avoid the potential confusion between the intended meaning of free meaning freedom and free meaning at no cost.

While FLOSS-licensed software may be developed in the same way as proprietary software (e.g., as in the case of MySQL), much of it is developed by teams of organizationally- and geographically-distributed developers, in what has been described as a community-based development [4], which is the focus of the research examined in this review. In some projects, a focal organization may take the lead in coordinating the efforts of a broader community of developers [5], but many projects exist outside of any formal organizational structure. Though recent years have seen an increase in the participation of firms in FLOSS and so in contribution from employees paid to work on FLOSS projects [6], even these contributions are often made available to the wider community [7]. As a result, FLOSS can be characterized as a privately-produced public good [8]. These private/public development practices are the focus of this review.

Over the past ten years, FLOSS has moved from an academic curiosity to a mainstream focus for research. There are now thousands of active FLOSS projects, spanning a wide range of applications. Due to their size, success and influence, the Linux operating system and the Apache Web Server and related projects are the most well known, but hundreds of others are in widespread use, including projects on Internet infrastructure (e.g., sendmail, bind), user applications (e.g., Mozilla Firefox, OpenOffice), programming language interpreters and compilers (e.g., Perl, Python, gcc), programming environments (e.g. Eclipse) and even enterprise systems (e.g., eGroupware, openCRX).

With this growth has come a concurrent increase in the volume of research examining the phenomenon. A review of this literature is important and timely for several reasons. First and foremost, FLOSS has become an important phenomenon to understand for its own sake. FLOSS is now a major social movement involving an estimated 800,000 programmers around the world [9] as well as a commercial phenomenon involving a myriad of software development firms, large and small, long-established and startup. On the user side, millions have grown to depend on FLOSS systems such as Linux, not to mention the Internet, itself heavily dependent on FLOSS tools. A recent report estimates that 87% of US businesses use FLOSS [10]. Ghosh [11] estimates the cost of recreating available FLOSS code at €12B, and notes "This code base has been doubling every 18-24 months over the past eight years, and this growth is projected to continue for several more years". As a result, FLOSS has become an integral part of the infrastructure of modern society, making it critical to understand more fully how it is developed.

As well, FLOSS represents a different approach to innovation in the software industry. The research literature on software development and on distributed work emphasizes the difficulties of distributed software development, but successful community-based FLOSS development presents an intriguing counter-example. Characterized by a globally distributed developer force, a rapid, reliable software development process and a diversity of tools to support distributed collaborative development, effective FLOSS development teams somehow profit from the advantages and overcome the challenges of distributed work [12]. FLOSS is also an increasingly important venue for students learning about software development, as it provides a unique environment in which learners can be quickly exposed to real-world innovation, while being empowered and encouraged to participate. For example, Google Summer of Code program offers student developers stipends to write code for FLOSS projects (http://code.google.com/soc/).

In addition to its intrinsic merits, FLOSS development has attracted great interest because it provides an accessible example of other phenomena of growing interest. For example, many researchers have turned to community-based FLOSS projects as examples of virtual work, as they are dynamic, self-organizing distributed teams comprising professionals, users and others working together in a loosely-coupled fashion [13, 14]. These teams are almost purely virtual teams in that community-based developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily by means of computer-mediated communications (CMC) [15, 16]. The teams have a high isolation index [17] in that many team members work on their own and in most cases for different organizations (or no organization at all). For most community-based FLOSS teams, distributed work is not an alternative to face-to-face: it is the only feasible mode of interaction. As a result, these teams depend on processes that span traditional boundaries of place and ownership. While these features place FLOSS teams toward the end of the continuum of virtual work arrangements [18], the emphasis on distributed work makes them useful as a research setting for isolating the implications of this organizational innovation. Traditional organizations have taken note of these successes and have sought ways of leveraging FLOSS methods for their own distributed teams, difficult without first understanding what these methods are.

Another important feature of the community-based FLOSS development process is that many developers contribute to projects as volunteers, without remuneration; others

are paid by their employers, but still not directly by the project. As a result, recruiting and retaining new contributors is a critical success factor for a FLOSS project. Furthermore, the threat of "forking" (starting a parallel project from the same code base), while uncommon and discouraged, limits the ability of project leaders to discipline members. These features make FLOSS teams extreme examples of self-organizing distributed teams, but they are not inconsistent with the conditions faced by many organizations when recruiting and motivating professionals or developing distributed teams. As Peter Drucker put it, "increasingly employees are going to be volunteers, because a knowledge worker has mobility and can go pretty much every place, and knows it… Businesses will have to learn to treat knowledge workers as volunteers" [19]. As a result, research on FLOSS development offers lessons for many organizations.

However, as Scacchi [20] noted, "little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success". While a great number of studies have been conducted on FLOSS, our review shows there have been few efforts made to integrate these findings into a coherent body of knowledge based on a systematic review of the literature. The few surveys that have been done [21-23] synthesize various major issues investigated in FLOSS research, based on a small set of studies, but without explaining how their review processes informed these issues or their sample strategies. Indeed, it is clear that the term FLOSS includes groups with a wide diversity of practices and varying degrees of effectiveness, but the dimensions of this space are still unclear. A key goal of our review is to synthesize the empirical literature to date in order to clarify what we know and do not know about FLOSS development and to suggest promising directions for further work.

This paper is organized as follows. The next section provides a brief overview of the research methodology applied in conducting this review. It is followed by two reviews of empirical studies that examine FLOSS development and use. Building on this review, we then identify trends as well as gaps in current research and provide suggestions for future research. The paper concludes with a summary of key points drawn from our review.

2 METHODOLOGY

This section describes the methodology we followed to identify and classify relevant work as a basis for a systematic review. Our goal in this paper is to summarize the findings of the published research on FLOSS, rather than presenting our own perspectives on the subject. Our literature review therefore required: 1) a literature search strategy; 2) the development of criteria for the types of studies to be included in our analysis; and 3) a coding scheme to analyze the selected studies. We performed two types of analysis, requiring two approaches to coding. The methods adopted for these tasks are described below.

2.1 Literature search strategy and criteria for inclusion

A challenge we faced in preparing this review is that the literature on FLOSS is expanding all the time, requiring a strategy for dealing with this growth while still producing a useful review article. To address this challenge we collected and analyzed papers for the review in two waves, the first in early 2006 and the second in early 2009.

At the time we began this review, FLOSS research was relatively new and often not published in journals, so we initially attempted to collect as many articles on FLOSS as possible before refining the collection. The first wave of papers was collected using three methods to search for appropriate literature, with a goal of identifying essentially all available papers on FLOSS to that point. First, we collected all papers from the opensource.mit.edu working paper repository (commonly used by researchers in the field to distribute papers), from journal special issues on FLOSS, and from FLOSS tracks in conferences such as the *International Conference on Open Source Software (OSS)* (organized by International Federation for Information Processing (IFIP) Working Group 2.13) and *International Conference on Software Engineering (ICSE)* workshops, as well as conferences in related fields such as the *Academy of Management* and *Association of Information Systems*. Second, we conducted a search in document databases such as ABI/Inform and Web of Science using "open source software" as the keyword (we had noted that most papers on FLOSS development used that term, and at the time even papers on free software included the term as a point of comparison). Finally, we looked through the reference lists of key articles to ensure that we had not overlooked other articles. The search process resulted in 586 papers in total. Given our goal of reviewing

FLOSS research to clarify what we know and what we do not know about FLOSS development, we limited our review to 138 published empirical studies where FLOSS development and usage were the main themes.

The dramatic increase in research after 2006 and increased acceptance of FLOSS as a research topic made it possible to focus our search in the second wave. The second wave of papers was collected only from journal articles published between 2006 and 2009, yielding a further 55 empirical papers. The resulting 193 papers are from 39 different journals and 41 different conferences, demonstrating the diversity of audiences for research on FLOSS development and highlighting the need for a systematic survey that pulls together work from these diverse sources, which would otherwise might go unnoticed by researchers working in a single discipline.

We carried out two types of analysis that are presented in this paper: a quantitative review and a qualitative review. The qualitative review, which is the main contribution of this paper, includes papers from both waves of collection, while the quantitative review includes only articles from the first wave for which the sample is complete. Papers from the second wave were not included in the quantitative analysis because of concerns that the selective collection of these papers had resulted in a biased sample, making quantitative summaries suspect. The approach taken for these analyses is described in the following subsections.

2.2 Quantitative review methodology

Th goal of the quantitative review was to provide an overview of the nature of research being published on FLOSS development. For this review, each article from the first wave was coded on several dimensions: publication year, publication venue type (conference or journal), level of analysis (e.g., group or individual), research methods (e.g., survey, case study), data collection methods, number and names of projects studied, reference disciplines that support the research, theories applied (if any) and the main constructs it examined. Specific categories for each dimension were developed by a group of coders until basic agreement was achieved on a sample group of papers. The full collection was then split between two coders working through an online system that showed their coding work, enabling coders to use codes created by other coders. The two coders met from time to time to review their use of codes. This information provides a quantitative

assessment of the state of FLOSS research as of 2006, as well as suggesting gaps and directions for future research.

2.3 Qualitative review methodology

The goal of the qualitative review was to identify constructs studied in the literature and to summarize research findings. A crucial task for a review paper is to provide a framework capable of organizing the existing literature and assisting future researchers in positioning their work in reference to that existing literature. We began our search for such a framework with a ground-up card sorting exercise. Four coders examined a sample of the literature from the first wave and inductively recorded codes for the concepts studied in the paper. These codes were used as the starting point for the systematic coding of constructs noted above. To develop the overall model, we then transferred these codes onto post-it notes and sorted them as a group on a white-board. We then used the results of this sorting process to guide a search for relevant frameworks in the literature, which in turn was used to structure the review of the papers from both waves of paper collection. The resulting research framework is discussed below in findings. Having identified the constructs studied in the literature and organized them in a framework, we then returned to the papers to identify the findings of each study, collecting together those that addressed similar constructs. These findings are presented below.

3 AN OVERVIEW OF THE FLOSS RESEARCH LITERATURE

In this subsection, we present the quantitative analysis of the research publications on FLOSS development. This analysis is based on the exhaustive survey of papers collected in the first wave, that is those published up through 2006. A little more than half of the sample, 55%, were papers from conferences; journal papers make up the remaining 45%. A sharp increase in the number of annual publication from 1999 through 2005 (Figure 1) demonstrates the increasing interest in the topic. This increase is reflected in the selective review of more recent publications. In particular, the increased acceptance of FLOSS research in journals allowed us to consider only journal publications in the second wave.

3.1 Level of analysis

FLOSS can be studied at different levels of analysis. We distinguished among studies at the individual, group, organization and societal levels. Approximately 8% of papers included multiple levels of analysis, most often integrating the group and organization levels. The literature demonstrates a strong preference in the literature for the group level of analysis, which makes up 57% of the studies in the sample, with an additional 17% at the individual level, 15% at the organization level, and just 4% at the level of society.

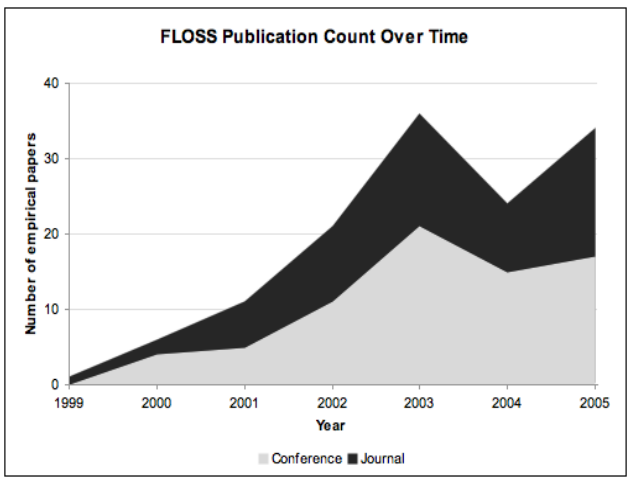

Figure 1: Annual counts of empirical research publications

3.2 Research methods

Although a variety of research methods were observed, the case study was the most common, making up 42% of all papers in the sample (n=57). Case studies were typically performed at the group level of analysis (n=31, 23% of total), and based on secondary data in more than half of these instances (n=17, 12.5% of total). It is notable, however, that all of the papers in the sample for which the coders found the data collection methods unclear were case studies. In addition, some case studies did not specify the number of projects in their sample, which implies that the details of data collection may frequently go unstated in case study research, while all other research methods had clearly identifiable data collection methods.

Surveys were the next most common choice of research method, appearing in about 25% of our sample. For these papers, it may be surprising to note that studies conducted at the group level were also based on secondary data in about two-thirds of the papers. This speaks to the overall trends for data types used in FLOSS studies; regardless of the data collection methods or source, 52% of studies in the sample were based on secondary data. Only about 10% of papers used interview data, and likewise, only about 10% used data collected through observation. Multi-method studies, while infrequent, were most likely to incorporate interviews with case studies, surveys and field studies.

3.3 Sample Size and Projects Studied

The distribution of sample sizes of projects studied in these papers was highly skewed toward single projects (42%), followed by studies of fewer than ten projects (18%), studies using repository data (16%) that may include anywhere from hundreds to thousands of projects, and studies of 10-100 projects (6%). (Repository data are data collected from FLOSS development sites such as SourceForge that include many projects.) Considering research methods and levels of analysis, the dominant form of FLOSS research overall is the study of a single project at the group level: 35 such papers comprise approximately 26% of our sample. This choice is closely related to the choices of research and data collection methods, as shown seen in Table 1.

With respect to the projects studied in the FLOSS literature, 42% of the papers sampled did not name the projects they studied, or did not study an open source project, but rather some other aspects of FLOSS, such as implementation. None of the studies using repository data named the projects that were studied, presumably due to the scale of the research (these studies can include 100s or 1000s of projects). The remaining 58% of the sample provide some interesting insights into the types of open source projects covered in the literature. The distribution of projects named in different studies shows a classic long-tail distribution, seen in Figure 2. Linux is clearly the most commonly studied FLOSS project, appearing in 30 studies, followed by Apache. Two-thirds of these papers studied only Linux and the remaining third of the papers included additional projects besides Linux. However, while Linux and Apache have been most studied overall, the frequency of papers including these two projects peaked in 2003 and dropped sharply thereafter. At the same time, the number of studies with no projects named, often

those examining a large sample of projects or using repository data (these factors correlate at r=0.98), have been increasing over time. This suggests that as data on a wider variety of projects became more easily available, the variety of projects studied also rose.

Table 1. Research methods and level of analysis

Research Methods	**Levels of Analysis**					
	Multi	**Society**	**Organization**	**Individual**	**Group**	**Total**
Case Study	4%	2%	5%	8%	23%	42%
Experiment					1%	1%
Field Study	1%	1%	1%	1%	3%	7%
Instrument Development			1%		3%	4%
Interview			1%		1%	2%
Multi	1%	1%	1%		1%	4%
Objects		1%	1%		8%	10%
Secondary	1%			1%	2%	4%
Simulation				1%	1%	2%
Survey	1%		4%	6%	13%	24%
Total	**8%**	**5%**	**14%**	**17%**	**56%**	**100%**

As indicated by the distribution of projects studied, shown in Figure 2, only 18 of the 51 projects (35%) named as subjects that appeared in our sample were included in more than one study. This trend brings into question how well the projects currently studied in FLOSS research represent the entire population; it is reasonable to expect that there are significant differences between Linux, for example, and such projects as VIM, GIMP, and XML included in other studies.

Figure 3 shows the tradeoff in FLOSS research between the sample sizes of projects studied and the intensity of the research approach. The size of the circle represents the relative number of the studies in that area. The figure shows the two types of studies that dominate current FLOSS research, as noted above: one or a small number of projects studied using the case study method or surveys covering a few variables to investigate

larger sample sizes. These two types of studies represent two extremes of the tradeoff between the depth and breadth of a research study, anchoring the ends of a frontier of feasible research designs indicated by the red arc in Figure 3.

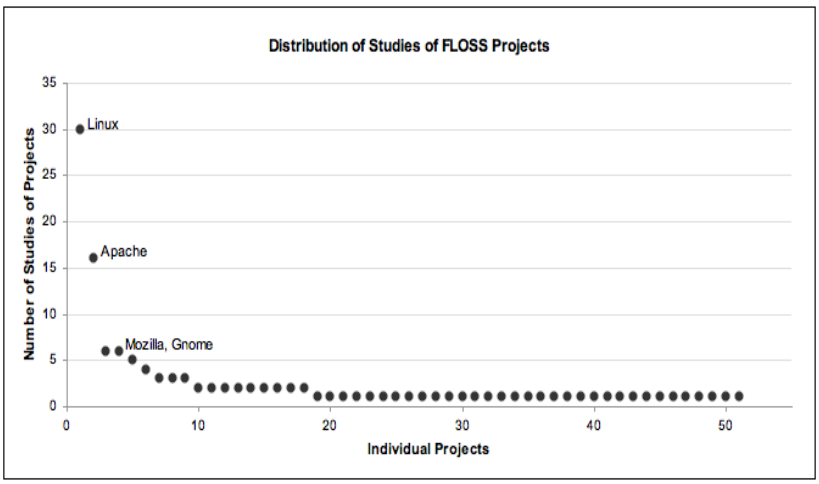

Figure 2. Distribution of Studies of FLOSS Projects

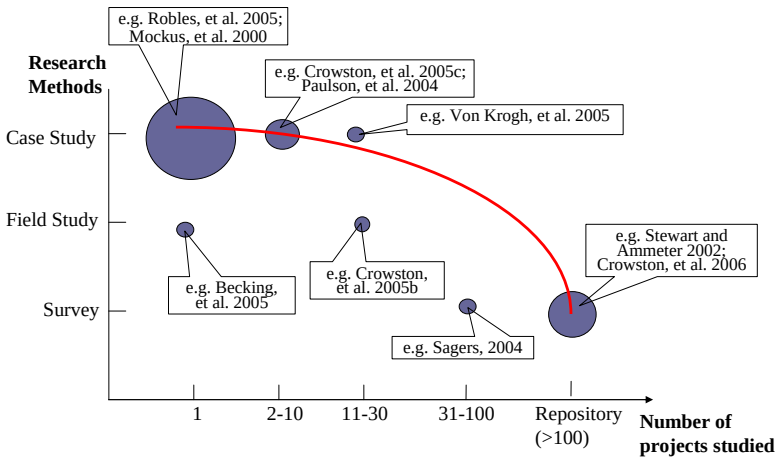

Figure 3. Distribution of Research by Research Methods and Number of Projects Studied

3.4 Reference Disciplines and Theories

We examined the reference disciplines and theories identified in the research papers to shed light on the intellectual underpinnings of these studies. Approximately 20% of the papers did not refer to any reference disciplines and about 64% referred to a single reference discipline. The remaining 16% of the papers incorporated two to four reference disciplines, with business and management influences present in 80% of these multidisciplinary papers. Business and management was also the most common reference discipline overall, with one-third of the total mentions. Computer science and computer engineering together comprise almost a third of the references as well, with information science and sociology being the next most common reference disciplines mentioned in the literature. In addition, 62% of papers that drew upon multiple reference disciplines employed theory, in contrast to 27% of papers that reference a single discipline, suggesting that the development and application of theory in this research area is often characterized by leveraging the intellectual resources of multiple disciplines.

We also examined how studies used existing theory. Case studies and surveys make up the bulk of the papers in our sample, and are also the most likely to contain references to theory, as seen in Figure 4: 35% of case studies mentioned theoretical content, as did about 44% of surveys. While only about 32% of the overall sample contained references to theory, the more technical research approaches of instrument development and studies of objects (usually code) had no instances of theory usage. This demonstrates one of the challenges in describing an interdisciplinary body of research literature, as not all studies' contributions can be adequately judged based on the traditional classifications that we have applied here.

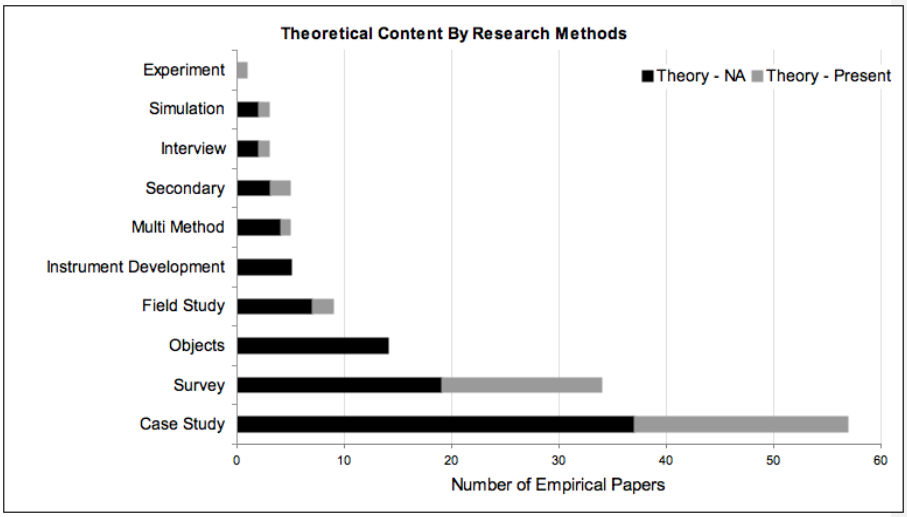

Figure 4. Inclusion of References to Theory by Research Methods

4 FINDINGS OF THE FLOSS RESEARCH LITERATURE

In this section, we present the main contribution of our review, namely an overview of the findings of the research literature, drawn from the published papers. In the first subsection, we provide a description of our organizing framework, which draws on an Inputs-Mediators-Outputs-Inputs (IMOI) model. This model is used to organize the constructs studied in the literature. A detailed review of the findings of the literature organized by these constructs follows in subsections 4.2–4.5, which forms the bulk of this paper.

4.1 An Organizing Framework for the Review

As noted above, we developed a framework for organizing the research papers on FLOSS development based on the constructs studied. We chose to organize these constructs according to the inputs-mediators-outputs-inputs (IMOI) model (Figure 5) [24], which draws together decades of work in the 'small group' literature [25-27]. This model most closely matched the inductive model and provided additional structure for the framework presented in this paper. We chose the IMOI model over earlier Input-Process-Output models [e.g., 26] because: 1) it distinguishes emergent states from processes, which

describe cognitive, motivational and affective states of a team, as opposed to the interdependent team activities; and 2) it provides feedback loops between outputs and inputs, treating outputs also as inputs to future team processes and emergent states [24]. The suitability of the model is unsurprising since most FLOSS development does occur in small teams and the majority of the studies conducted research at the project level of analysis. Where necessary, we adapted the model to incorporate detailed constructs directly tied to the software engineering context of FLOSS work.

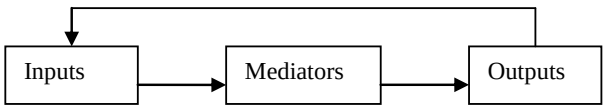

Figure 5. Inputs-Mediators-Outputs-Inputs Model (adapted from Ilgen et al. 2005)

Figure 6 shows the resulting framework, with the major concepts that we identified in the FLOSS research papers in each of the categories of the IMOI model. Inputs represent starting conditions of a team, such as its member characteristics and project/task characteristics. Mediators represent variables that have mediating influences on the impact of inputs on outputs. Mediators can be further divided into two categories: processes and emergent states. Processes represent dynamic interactions among team members as they work on their projects, leading to the outputs. Emergent states are constructs that "characterize properties of the team that are typically dynamic in nature and vary as a function of team context, inputs, processes and outcomes" [27, p.357]. Outputs represent task and non-task consequences of a team functioning [28].

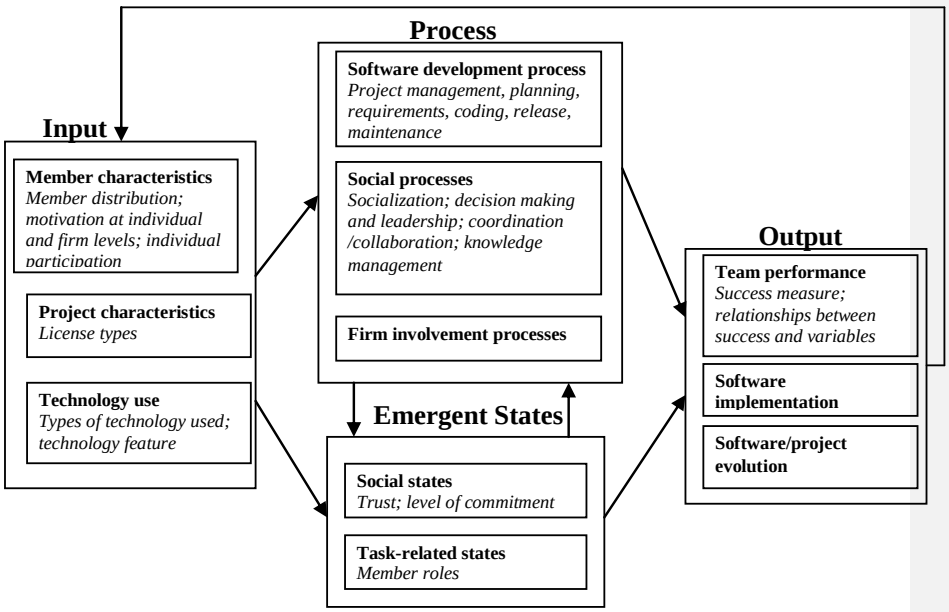

Figure 6. Constructs Studied in the Reviewed FLOSS Research Papers and Their Relations

A variety of constructs were observed in the literature, with one to seven distinct constructs identified in each paper. The most commonly studied class of construct is project characteristics, which makes up 21% of all instances of constructs studied in the first wave, indicating the descriptive nature of much of the FLOSS literature in our sample. Project characteristics are overwhelmingly studied through secondary data (15% of total), while constructs such as implementation and context do not rely as heavily on a single data type. Social processes (16%) and success (12%) were the next most frequent constructs observed, and studies of these constructs were also strongly reliant on secondary data. In contrast, studies of motivation tended to use questionnaire data more often than other types of data.

Certain research methods are also more strongly aligned with certain constructs; for example, field studies are most often used with the construct of social processes in the first wave, and instrument development is most frequently related to research methodology. The level of analysis is also relevant to the constructs studied. As we have mentioned, the overwhelming majority of studies are at the group level. However, not all constructs conform to this trend. Motivation is more often studied at the individual level

and implementation was most frequently studied at the organizational level, while tasks were studied at multiple levels of analysis. We now present our review of the literature, drawing on papers from our full collection, using this framework to structure our discussion.

4.2 Inputs

We first discuss papers that analyzed an input to the FLOSS project process. Inputs represent the design and compositional characteristics of a team, such as members' skills and personalities, group size and composition, technology use and task characteristics that influence how the teams work [28, 29]. Inputs that have been investigated by previous FLOSS research can be grouped under the labels of member characteristics, project characteristics, technology use and context.

4.2.1 Member Characteristics. The member characteristics of FLOSS teams have been mainly examined with respect to geographical location, motivations at the individual level, participation at the individual level, and firm-led participation in FLOSS development.

Geographic Location. There have been a number of studies of the geographical location of FLOSS participants. By using self-reported descriptions of developer activity containing geographical information that are by-products of project practices, these studies claim that FLOSS involves global development efforts, dominated by North American and European participants. For example, by examining the Linux CREDITS file on all major kernel releases and developer contact information on the GNOME project website, Lancashire [30] showed a predominance of developers in the United States and Northern Europe in raw numbers of contributors. After adjusting for population size and prevalence of Internet access, the U.S. declined in influence (high absolute numbers but low relative numbers) while Europe, especially Northern Europe, grew in importance. By tracing individual-level information such as email addresses and time zone information from SourceForge database and the mailing list archives of Debian, GNOME and FreeBSD projects, Gonzalez-Barahona, et al. [31] reported that North America and Europe (especially German, UK and France) are the top regions for FLOSS developers. Further, European developers have overtaken North American developers in terms of active participation in recent years. Other studies have confirmed the relatively

high representation of European FLOSS participants, particularly from Germany and Finland [32, 33].

Motivation for participation. FLOSS teams are nothing without contributing participants and the question of what motivates participation has been a central theme in studies of FLOSS. Much of the empirical work in this area is driven, at least rhetorically, by a reaction to early analytical modeling articles by economists (e.g. [34, 35]) who argued that motivation was derived from indirect signaling about quality, with the payoff to come in higher career earnings. While it would be premature to argue that this work has been discredited, the empirical work on motivations has found little evidence for these expected motivations. In general, this body of research has addressed individuals' motivations for joining FLOSS development, as well as those of firms.

Motivation at individual level. Most research has focused on individual motivations. Early empirical work on this topic documented a range of factors that propel individuals to contribute to FLOSS projects, with consistent results. These studies found that motivations are heterogeneous, and generally there are three types: extrinsic motivations, intrinsic motivations and internalized extrinsic motivations. Our analysis revealed that reputation [36] and reward motives such as career development [37-39] are the two most frequently mentioned extrinsic motivations. Enjoyment-based motivations such as fun [40] and sharing or learning opportunities [41, 42] are the two most commonly mentioned intrinsic motivations. User needs [34, 43] is the most commonly mentioned internalized extrinsic motivations.

Recently, researchers have advanced this line of research by exploring what motivates individuals to continue participating in FLOSS [e.g. 42, 44, 45]. Results show that individuals' motives are not static, but evolve over time. For example, based on interviews and archive data analysis of a FLOSS project hosted on SourceForge, Shah [42] found that a need for software development drives the initial participation, but the majority of participants leave once their needs are met. For the remaining developers other motives evolve, and participation may become a hobby. By studying OpenOffice.org, Freeman [46] argued that individuals' motivations to join and continue participate in the FLOSS projects are related to personal history prior to and during participation. In the PhpMyAdmin project, Fang and Neufeld [44] revealed that initial motivations to participate did not effectively predict long-term participation, but situated

learning and identity construction behaviors were positively linked to sustained participation.

A few studies have gone beyond reports of motives to examine how intrinsic, extrinsic and other factors interact to influence their participation in particular projects. [e.g. 47, 48]. For examples, by studying 135 projects on SourceForge, Xu et al. [49] found that individuals' involvement in FLOSS projects depends on both intrinsic motivations (i.e. personal needs, reputation, skill gaining benefits and fun in coding) and project community factors (i.e. leadership effectiveness, interpersonal relationship and community ideology). Through their comparison of FLOSS developers in North American, China and India, Subramanyam and Xia [50] found that developers in different regions with similar project preferences were driven by different motivations. For instance, for projects that are larger in scale, more modular and universal in nature, Chinese developers are drawn by intrinsic motives while Indian developers are found to be mostly motivated by extrinsic motives.

Motivation at firm level. At present, more and more software companies are being involved in open source software development, primarily through creating a service business around open source software, or by sponsoring open source software and employing software engineers to participate [51, 52]. Surveys have shown that as many as 45% of contributors are paid by firms for their participation, either directly or indirectly [38]. Research on this topic has also examined the reasons that companies are investing internal resources in FLOSS development. For example, Bonaccorsi and Rossi [53] found that firms are motivated to be involved with FLOSS because it allows smaller firms to innovate, because "many eyes" assist them in software development, and because of the quality and reliability of FLOSS, with the ideological fight for free software coming at the bottom of the list. In comparison with individuals they find that firms focus less on social motivations such as reputation and learning benefits. Similarly, by studying the firm-developed innovations embedded within Linux, Henkel [54] emphasized the importance of receiving outside technical support as a motivator to revealing code.

コメント **[AW1]:** It see
that most firms are not g
software ideology much
could probably drop this

Individual participation. A few studies have examined reported individual measures of time commitment, using either self-reports or attempts to impute total time from public records (such as lines of code contributed or mailing list messages sent). Luthiger Stoll

[55] found that participants spend an average of 12.15 hours per week, with project leaders averaging 14.13 hours, and bug-fixers and otherwise active users at around 5 hours per week. Lakhani and von Hippel [56] studied the amount of time participants spend reading and answering user support questions in the Apache forums, finding that the most frequent answer providers spent 1.5 hours per week, but that frequent information seekers spent just half an hour.

In addition to actual time spent, researchers have examined individual tenure with projects, or the length of time that participants continue to participate. Howison et al. [57], which studied 120 relatively successful SourceForge projects, found that the most common length of participation, across all roles, was no longer than a single month. The tenure of participants varies significantly according to their role. For example, Robles and Gonzalez-Barahona [58] found comparatively long tenure amongst Debian package maintainers (more than half of the maintainers in 1998 continued to maintain their packages in 2005.)

More recent work has explored the factors that influence individuals' level of contribution. In Roberts et al. [47] studied the impact of different motivations on individual contribution levels in Apache project. The results showed developers' paid participation and status motivations lead to above-average contribution levels; use-value motivations lead to below-average contribution levels; and intrinsic motivations do not significantly impact average contribution levels. In another study, Fershtman and Gandal [59] studied the relationship between licenses types and individuals' contribution level. They found that the output per contributor in open source projects is much higher when licenses are less restrictive.

4.2.2 Project characteristics. Another input variable that is often examined in the FLOSS literature is project characteristics, that is, the distinguishing features of these projects. Software license types attract the most attention in this topic as a particularly concrete differentiator of FLOSS projects. Licensing is also one of the most important tactics that used by a project to allow its intellectual property to be publicly and freely accessible and yet governable [60].

The license type used in a project has been identified as playing a crucial role with respect to all activities in FLOSS development, such as motivations, coordination, and relationships with commercial firms [22]. Two commonly investigated features of a

FLOSS license are inclusion of so-called copyleft provisions (requiring that modified versions of the software also be open) and viral provisions (requiring that the modified versions of the software be combined only with other programs distributed under licenses that share the first requirement) [61]. Though there is a clear focus in conceptual work on this topic, a few empirical studies have been conducted to examine the influence of license choices on various aspects of FLOSS development. By examining the SourceForge projects, Lerner and Tirole [62] concluded that "projects geared toward end-users tend to have restrictive licenses [i.e., with copyleft and viral provisions], while those oriented toward developers are less likely to do so. Projects that are designed to run on commercial operating systems and whose primary language is English are less likely to have restrictive licenses. Projects that are likely to be attractive to consumers—such as games—and software developed in a corporate setting are more likely to have restrictive licenses. Projects with unrestricted licenses attract more contributors." (p.20). Using data gathered from the Freshmeat website (www.freshmeat.net), Stewart and her colleagues found that OSS projects that used a non-restrictive license became more popular over time than those that use a restrictive license. On the other hand, using data from 62 projects in SourceForge (www.sourceforge.net), Colazo et al. [63] found that copyleft licenses were associated with more successful projects in terms of higher developer membership and productivity.

4.2.3 Technology Use. The type of technology used by FLOSS teams is an important input since FLOSS team members coordinate their activity primarily by means of computer-mediated communications. But surprisingly little research has examined the use of different software development tools and their impact on FLOSS team activities. One exception is Scacchi [64], who discusses the importance of software version control systems such as CVS or Subversion, both for coordinating development and for mediating control over source code development when multiple developers may be working on any given portion of the code at once. This paper also discusses the interrelation of CVS use and email use (i.e. developers checking code into the CVS repository discuss the patches via email). Michlmayr [65] illustrates the importance of bug trackers to coordinate among those working on questions and answers.

コメント **[AW2]:** If we revision on this, includin paper from OSS 2008 on be great for this section.

A small body of research studies the tools developers or users use to share and exchange knowledge. For example, Robbins [66] discusses nine types of commonly used OSS engineering tools and illustrates their impact on the software development process. Using data from the developer mailing lists of two open-source software projects, Lanzara and Morner [67] argue that technological artifacts and software-based artifacts are critical for knowledge sharing and creation in OSS development.

4.3 Processes

Processes are the dynamic interactions among FLOSS team members as they work on a project. Research on processes in FLOSS development has focused on software development practices and social processes within the projects. Increasing firm involvement in FLOSS development is leading researchers to investigate how firms make use of FLOSS to gain profits. In the following section, we review the empirical findings related to these three themes.

4.3.1 Software development practices. In this section, we review the findings of research on the practices followed by FLOSS teams for software development. Researchers have suggested that FLOSS development does not seem to readily adopt modern software engineering processes [64]. Projects usually rely on "virtual project management", meaning that different people take on management tasks as needed. In this way, the project also mobilizes use of private resources [64]. In the following sections, we consider the research on more specific practices, using the systems development lifecycle as an organizing structure.

コメント [AW3]: refere

Planning. It is commonly held that FLOSS projects do not engage in formal planning. For example, Glance [68] examined the kernel change logs to determine the criteria applied for releasing the Linux kernel. She argued that a release contained whatever had been added, as opposed to a clear process of planning the functionality needed for a release. However, projects often do have TODO lists or feature requests that form a sort of agenda for development [69]. Planning seems to be one contribution made by firms involved with projects [5].

Software Requirements Analysis. Similar to the planning stage, FLOSS projects are often said to not conduct formal software requirements analyses. Scacchi [64] states that FLOSS projects do not have conventional requirements documents. Instead, requirements

are found in email messages, which emerge from discussions among users and developers about what the software should and should not do, and from after-the-fact assertions by developers about the need for a new piece of functionality. Similarly, Mockus et al. [70] state that a user community can communicate its needs through bug reports or feature requests. Again, this is an area in which firm involvement may lead to changes [5].

Coding. Much work in this area focuses on modularity and software architecture. Modularity has been seen as key to the feasibility of distributed development. Scacchi [64] notes the importance of what he called "software extension mechanisms" that allow developers to easily add functionality to FLOSS projects via scripting or plug-ins. MacCormack, et al. [71] reported that a redesign of Mozilla resulted in an architecture that is much more modular that its predecessors, which emphasizes the important role of purposeful managerial actions in increasing modularity.

Testing. There are mixed results reported by the research on testing processes in FLOSS development, which vary by projects. Glance [68] notes that the Linux kernel release process was not based on formal testing. Rather, it relies on individuals testing their own code before submission and subsequent testing by users of releases, also described as peer review [68]. Stark [72] notes that peer review has been used in conventional development as well and cites research showing that it works without meetings. Although this survey shows that only half of 23 FLOSS respondents said that their code was reviewed, one possible explanation is that it might be reviewed later, since it is open to inspection by any interested party. It also notes that any quality control approach relies on developer commitment to the process, which may come from compliance, identification or internalization, suggesting FLOSS relies on later mechanisms. Other studies have shown that some projects have more formal testing for releases. For example, Thomas [73] describes several testing processes, such as the Linux Test Project, the Linux Stabilization Project. Dinh-Trong and Bieman [74] notes that FreeBSD does have a more defined testing process.

Release. The nature of open code is that users can always access the latest version, which may or may not be stable, so it is difficult to speak of a single FLOSS approach to releases. Erenkrantz [75] compared release practices of Apache httpd, Subversion and

Linux on dimensions of release authority, versioning, prerelease testing, approval of releases, distribution, and formats, and noted considerable differences among the projects.

Some projects have quite informal release schedules, following advice from Raymond [16] to "release early; release often". For example, Glance [68] found that in Linux, releases came at an irregular rate. It is not clear what drove the schedule, but a release pattern was observed in terms of accepting patches for a while, then freezing acceptance of new code to allow the system to stabilize, though stability was assessed only by an absence of reported problems. In contrast, some projects have more organized ways of releasing. Dinh-Trong and Bieman [74] report that FreeBSD releases a new version every four months. A "Release Engineering Team" coordinates the release, following a pattern similar to that of Linux. Projects tend to release incrementally, which suggests that making releases every six months (to allow time to control bugs) may be too infrequent.

Maintenance. Maintenance is a primary activity in FLOSS development, as in conventional development. In FLOSS development, however, the nature of maintenance is more like reinvention, which acts as "a continually emerging source of adaptation, learning, and improvement in FLOSS functionality and quality" [64]. Studies of maintenance in FLOSS has focused on activities such as problem solving processes, user support practices [56], change cycles (bugs and new features), software quality maintenance work, improvement, bug fixing processes, problem resolution interval, patches (internal or external submission), shallow bugs, and incident/problem/change management. Maintenance has been done differently in different projects. For example, in Linux, user support is often done commercially [76]. Other projects have commercial sponsors who sell support (MySQL, SugarCRM). Smaller projects tend to rely on community support, e.g., via email or discussion boards. However, Singh et al. [77] analyzed help interactions and found that this process was often inefficient because initial posts lacked the necessary information to answer questions, resulting in back-and-forth postings. The authors suggested that some details be captured automatically in initial reports, and also articulated the potential benefit of developing practices and tools for more explicitly reusing information, e.g., marking particularly helpful answers to automatically populate a kind of FAQ.

4.3.2 Social Processes. Social processes capture cognitive, verbal and behavioral activities performed by team members to manage interpersonal relationships among them

[27]. To date, the majority of FLOSS research pertaining to social processes has focused on socialization, decision making and leadership, coordination and collaboration, and knowledge management.

Socialization. The work on motivations shows that there is a large pool of people with motivations sufficient to participate in FLOSS development. Yet this number is substantially smaller than the number of active users of software and, presumably, smaller than the number of people who have ever considered participating in an open source project. The process of moving from a non-participant to a fully-fledged FLOSS developer has been addressed in a small volume of literature on socialization in FLOSS projects, which examines the strategies and processes through which new members join an existing FLOSS development community. This body of literature treats socialization as a process that places emphasis on a participant's actions and willingness to understand not just the code base but also the social structure of the project.

For example, in a study of socialization in the Freenet project, von Krogh et al. [78] propose that joining script (the level and type of activity a joiner goes through to become a member of the development process), specialization of new members, contribution barriers, and the feature gifts a new member can contribute are related to the process of being a new member. Confirming the findings of von Krogh et al. [78], Duchenaut [79] studied socialization in the Python project from both learning and political perspectives, and found that participants who move to the center of a project, acting in a way that exposes more of the network to them, come to understand the relationships between people and code and, largely through action in the form of code, or detailed discussions of code, build legitimacy and "enroll allies" for their evolution towards the core of the project. He also highlights the manner in which the onus for socialization falls almost entirely on the developer, rather than the team. The process thus acts as a filter for participants that match the project, removing the need for recruitment effort typically required in the software industry.

Decision Making and Leadership. In conventional teams, decision-making effectiveness is very important to team effectiveness. A lack of transparency and consideration in the decision making process tends to alienate those who are not being consulted and erodes the sense of community [80].

One common concern in studies of FLOSS teams' decision making is decision style, which depends on the hierarchy of the developers. As Gacek and Arief [81] pointed out, a stricter hierarchy differentiates between levels of developers and generates a more centralized power structure, while a looser hierarchy treats developers on a similar level and implies a decentralized decision-making process. Both centralized and decentralized decision making styles have been examined. Shaikh and Cornford [82] examined how debate over version management tools (CVS versus BK) reflects governance and decision making processes in the Linux Kernel community, providing an example of a centralized decision making process. Moon and Sproull [83] also pointed out that in Linux, Linus Torvalds originally made most of the key decisions for the team. German [84] provides a decentralized decision making example by studying the GNOME project. Committees and task forces composed of volunteers are created to complete important tasks. Annual face-to-face meetings are also organized to make major decisions. By doing so, GNOME flattens the organizational structure of the project and allows broader participation in the decision-making process. In the Apache web server project, members adopt the voting mechanism to reach consensus [85]. Researchers have also noted that decision making styles might change over the life of the project. In the early life of a project, a small group will control decision making, but as the project grows, more developers will get involved [5].

Closely related to decision-making, leadership has been much discussed in the literature. The main duties of a leader in FLOSS projects includes providing a vision; coordinating contributors' efforts; attracting developers to the project; and keeping the project together and preventing forking.[34, 86] Research has focused on who can become a leader in FLOSS development teams. First, leaders are usually not appointed, and in most cases not formally identified, but rather emerge from participation in FLOSS development. Individuals are perceived by others as leaders based on their sustained and strong technical contribution [87, 88], diversified skills [86] and a structural position in their teams [88, 89]. Second, FLOSS teams usually exhibit shared leadership instead of having a single leader [90]. According to Fielding [91], shared leadership enables these teams to continue to survive independent of individuals, and enables them to succeed in a globally distributed and volunteer organizational environment. Similarly, Mateos-Carcia

and Steinmueller [92] reported that the distribution of authority and decentralization found in the Debian community facilitate its growth and development.

Coordination and collaboration. Collaboration occurs through coordination, which manages dependencies between activities [93]. The FLOSS environment makes coordination more difficult for several reasons. Volunteers without formal contracts, geographically and temporally dispersed contributors, the virtual environment, and different types of actors (firm-sponsored vs. volunteers) are factors that all complicate coordination efforts [94]. Coordination activities play an important role in FLOSS development, and is critical to project success [95] and to sustaining collective efforts in FLOSS teams, especially for large project such as Linux [96].

The backgrounds and characteristics of the different projects may influence the use of coordination mechanisms in general. For example, Java tends to uses ex ante mechanisms (i.e., coordination before taking action) while Linux tends to uses ex post mechanisms (i.e., coordination after action), which researchers suggest is because Java is a company-sponsored project while Linux is a community project [69, 94]. Using 10 large FLOSS projects, den Besten, et al. [97] found that collaboration effort was associated with the complexity of the code.

The literature review reveals four types of coordination mechanisms that are frequently discussed in FLOSS development:

Mechanisms to control the number of developers. The general collaborative mode of FLOSS development is that a small portion of developers are responsible for most of the output [70, 74, 98, 99]. Based on a theoretical framework of network governance, Sagers [95] demonstrates that restricted access to the development team improves coordination within the project.

Modularity and division of labor. Modularity is the most explicit mechanism used in FLOSS development. It keeps the core software product small enough to be handled by a small core group, and makes communication smooth and effective [70, 80]. An example is Linux, as Dafermos [100] stated, "modularity makes Linux an extremely flexible system and propels massive development parallelism and decreases the total need for coordination". However, Mockus et al. [70] notes that while developers tend to work on the same modules repeatedly, most modules were worked on by several people, which does not support the notion of individual code ownership and requires other ways of

coordinating. One possible way is to introduce coordinators to coordinate development between modules, as suggested by Asklund and Bendix [101].

Task assignment mechanisms. Findings on task assignment mechanisms across the literature are quite consistent. Contrary to commercial software development, self assignment is observed as the most common mechanism used in FLOSS development [70, 102-105].

Instructive materials and standardization initiatives. Instructive materials and standardization initiatives are another means used to coordinate software development effort. Instructive materials include guidelines for writing software and policies that enable developers to work independently; standardization initiatives standardize the software specifications to increase convergence between different files [80, 94].

In addition to these coordination mechanisms, teams need mechanisms to manage conflict. From interviews, van Wendel de Jooode [106] identified four conflict management mechanisms between firm-supported developers and voluntary developers: third-party intervention, modularity, parallel software development lines, and the exit option.

Knowledge management. There is a growing body of research recognizing that OSS development faces knowledge management (KM) challenges because of its highly distributed, knowledge intensive characteristics [107-109]. This body of research focuses on how knowledge is shared or reused in FLOSS development [56, 67, 77, 100, 110-113]. For example, Huysman and Lin [110] find that online communities without strict membership requirements activate cross-boundary learning and knowledge sharing Based on the analysis of developer mailing lists of two large-scale open source projects, Lanzara and Morner [67] illustrate how the processes of knowledge making and sharing are supported by dense social interaction and by the peculiar organizing features inscribed in technological artifacts. Von Krogh et al. [113] report on the reuse of knowledge in software development based on 15 open source projects. The authors find that the effort to search, integrate and maintain external knowledge influences the form of knowledge to be reused. Using 128 discussion threads from K Desktop Environment (KDE) mailing list, Kuk [114] reported positive relationships between conversational interactivity, cross-thread connectivity and knowledge sharing, and a curvilinear relationship between participation inequality and knowledge sharing.

コメント [AW4]: Whic

Learning theory provides a common theoretical perspective for this work, which frequently draws on communities of practice literature to conceptualize how knowledge is created and shared online. Using the case of the Linux kernel development project, Lee and Cole [111] described how the product development process can be effectively organized as an evolutionary process of learning driven by criticism and error correction. Hemetsberger and Reinhardt [112] take a social view of learning and knowledge creation to demonstrate how online communities of practice successfully overcome the problem of tacit knowledge transformation through technological tools, task-related features, collective reflection, stories and usage scenarios.

4.3.2 Firm involvement practices. The success of FLOSS has attracted more firms interested in profiting from FLOSS development; however, research on this aspect of FLOSS involvement is limited. Correspondingly, researchers are now investigating the processes of how firms make use of FLOSS, or the FLOSS commercialization process [115, 116]. Research has revealed that firms usually adopt a hybrid production model by combining proprietary software and open source software models [116]. The strategies that firms use to create new or utilize existing FLOSS communities are also discussed [117]. For example, by studying four firms involved with FLOSS, [118] discovered three ways firms used to connect with FLOSS communities: accessing development in the community in order to extend their resource base; aligning their strategy with the work in the community; and assimilating the work from the community.

4.4 Emergent States

In this section we review research that has examined moderators between inputs and outputs in the form of emergent states of the FLOSS project teams.

4.4.1 Trust. We first examine research that considers project team trust. Trust has been studied extensively in small groups research, and has been noted as a determining factor to achieve the effectiveness of team collaboration. Researchers have also suggested that trust is important in FLOSS team development [89, 119]. Trust is often related to team effectiveness. For example, Stewart and Gosain [119] propose that shared ideology enables the development of affective and cognitive trust, which in turn lead to group efficacy and effectiveness. In a study of leadership in FLOSS teams, Fleming and Waguespack [87] argue that successful leaders are the product of strong technical

contribution and a structural position (brokerage and boundary spanning) that can bind the community together. But the effectiveness of brokerage is contingent on trust. An inherent lack of trust associated with brokerage positions can be overcome through physical interaction or contributions within technological boundaries. But not all researchers share the same belief. In a study of published case studies of FLOSS projects, Gallivan [120] found that group effectiveness can be achieved in the absence of trust if a set of control and self control mechanisms is presented.

4.4.2 Task Related Structures. We next consider task related social structures, including roles, level of commitment and shared mental models.

Roles. In an emergent context, something as seemingly simple as role becomes more complex. While in one context and one time, a participant may be a 'core developer', in another context they may be a support question asker [41]. Most research on roles focuses on the differences between distinct roles and how to define roles. Gacek and Arief [81] suggest distinguishing between passive and active users, and among active users, between non-developers and developers, and among developers between co-developers and core developers, with corresponding increases in responsibility and contribution.

Alternative methods have been used to examine the sizes of the core and periphery groups. Crowston et al. [57] studied the distribution of contributions to the bug-tracking system, for 120 Sourceforge teams using three different methods: first, the self-reporting of teams based on the list of developers on the Sourceforge site; the second, core/periphery measures using the social network structure; and third, an analogy to Bradford's law about the distribution of academic publications. The measures indicated that the developer's list was not a good indicator of core membership (at least in bug-fixing) and that the skew of contribution in the communications domain was substantially higher than in the code domain. The more reliable measures pegged the core group size at a median of 3 (about 5% of participants in the project). The results are basically in line with early results in sociology on feasible group sizees, such as James [121].

Lin [122] describes interviews she conducted with firms involved in open source and with developers that had moved from community involvement to working for a company, but still doing open source. She found that developers working inside companies have

hybrid roles, such that they can draw on resources from the firm and the community, but have to balance their loyalties and act as translators in situations of different aims.

Level of Commitment. Researchers have also been interested in the distribution of different types of effort, such as code contribution [70, 102], communication contribution [123] and support contribution [56]. Not all development teams and community members contribute equally, and the ratio of contributions has become a frequent question of interest in empirical studies of FLOSS development. In a study of the Apache community, Mockus et al [70] observed that the top 15 contributors (out of 388 total) had contributed over 83% of modification requests and 66% of problem reports, which is lower but still quite high. They compared these contribution distributions to commercial projects and found them to be higher, sometimes substantially so, suggesting that while FLOSS projects have larger total numbers of contributors, the bulk of activity, especially for new features, is quite highly centralized. Efforts to replicate these findings have tended to show a smaller differential in the distributions. Dinh-Trong [74], in a study of the FreeBSD project, found that the top 15 contributors (of 354) contributed only 57% of new source code and one needed the top 47 to reach the 80% figure. Bug fixing is again found to be more widely distributed, with the top 15 checking in only 40% of the changes. They observed that these statistics cumulate effort over the entire lifetime of the project and so recalculated the measures within a three-year windows, but still found that the core group for FreeBSD is larger than that of Apache's. Koch and Schneider [98], studying the GNOME project, also found lower skew (top 15 contributing only 48%) but argue that there is still evidence for a small, more active, core group.

Research has only scratched the surface of the context of individual participation in FLOSS projects. For example, given that many participants work on projects as volunteers, it follows that work on a particular project, or on FLOSS projects in general, is only one among many activities the individual pursues, and not normally the main activity. This observation seems axiomatic in the case of volunteers but is shared even amongst those who are paid for activities relating to their participation. Fielding [91] relates that all the core participants in Apache had other “real jobs”, which usually involved using the software for which they were a contributor. Lakhani and von Hippel [56] finds that Apache participants spend about 30% of their work time on web servers.

Luthiger Stoll [55] surveyed developers about the balance between work time and spare time. The author found that over 60% of the time spent contributing was considered spare time by participants and those who consider they have more spare time are likely to spend more of it developing FLOSS, although the strength of the relationship fell as spare time continued to rise (decreasing returns). This finding is supported by Michlmayr [65], who reported that participants understand others to have "real" jobs that justifiably interfere with their participation. In addition, Crowston et al. [124] report that participants at face-to-face conferences cite the ability to spend long blocks of relatively uninterrupted time focusing on a FLOSS project as a major benefit of attendance.

Shared Mental Models. Prior research suggests that the existence of accurate shared mental models that guide member actions are important for team effectiveness [125]. Research on software development in particular has identified the importance of shared understanding in the area of software development. Curtis et al. [126], note that, "a fundamental problem in building large systems is the development of a common understanding of the requirements and design across the project team."(p.52) They go on to say that, "the transcripts of team meetings reveal the large amounts of time designers spend trying to develop a shared model of the design" (p.52). Scozzi et al. [127] analyzed mental models in a FLOSS project using cognitive mapping and process analysis. Specifically, they compared the mental models of four developers from the Apache Lucene Java project. Their analysis suggests that there is a high level of sharing among core developers on aspects such as key definitions (e.g., project goals, users and challenges) and some aspects of the causal maps, but the sharing was not complete, with some differences related to tenure and role in the project.

4.5 Outputs

Finally, we consider research that has examined the outputs of FLOSS project teams. Outputs represent task and non-task consequences of a FLOSS team's efforts or the outcomes of FLOSS implementation. Three recurring themes were observed in the research of FLOSS output: 1) the performance (i.e. effectiveness/success) of the team; 2) open source software implementation; and 3) evolution of the software and the project.

4.5.1 FLOSS Team Performance. We classify this body of research into two themes: 1) measures of FLOSS success and 2) relationships between performance and other variables.

Measures of FLOSS team/project success. Success is one of the most frequently used dependent variables in information systems research. So it is necessary to understand how previous research assesses the success of FLOSS projects. Several measures have been proposed. Based on a combination of literature review of IS field, a consideration of the OSS development process, and an analysis of the OSS developers' opinions, Crowston et al. [128] identified 7 measures of FLOSS project success: system and information quality, user satisfaction, use, individual and organizational impacts, project output, process and outcomes for project members. Similarly, Lee, et al. [129] proposed five measures of FLOSS success based on Information Systems (IS) literature: software quality, use, user satisfaction, individual net benefits and community service quality. These measures indicate that FLOSS success is a multidimensional construct, but most empirical research has only used one of these dimensions to assess success.

The most frequently used measure of success emerging from these studies is system and information quality. Although theory has described indicators such as code quality and documentation quality to measure FLOSS system and information quality [128], the majority of the empirical work uses code quality measures. This body of literature provides a variety of indicators to measure code quality such as maintainability [130-134], product/software quality [65, 68, 135-137], defect density [102, 138], usability [137, 139, 140], reliability [76, 136, 141, 142], and value of software output [142].

Relationship between success and other variables. More frequently, research focuses on exploring the relationship between success and its antecedent variables. Various factors have been examined for their impact on project effectiveness, typically focusing on specific project characteristics such as software components, team size, project types, project life cycles, sponsorships and license types. For example, based on longitudinal data on FLOSS projects hosted on SourceForge, Subramaniam et al. [143] found that restrictive licenses have an adverse impact on FLOSS success. Similarly, based on data from 62 relevant OSS projects, Colazo et al. [63] found that copyleft projects were associated with higher developer membership and productivity. In another study of 240 open source projects registered on Freshmeat, Stewart and Ammeter [144]

コメント **[AW5]:** This t
elsewhere to better effect
removing the duplicate u

found that sponsorship of a project, project types, and project development status are all related to one measure of project success: popularity (i.e. how much user attention is focused on the project).

Some studies reported the impact of different processes on team effectiveness, including knowledge sharing [145], network embeddedness [146] and leadership [147]. For example, base on 75 FLOSS projects, [148] reported a high degree of openness in governance practices lead to higher software quality.

A few studies investigate the impact of emergent state factors on project performance. For example, using content analysis to examine a set of published case studies of OSS projects, Gallivan [120] noted that although trust is rarely mentioned, ensuring control is an important criterion for effective performance within OSS projects. Wynn [149] found the fit between the life cycle stage and the specific organizational characteristics of these projects (focus, division of labor, role of the leader, level of commitment, and coordination/control) is an indicator of the success of a project as measured by the satisfaction and involvement of both developers and users.

Further, some studies compare the quality of open source software with propriety software and the results are mixed. For example, by comparing three closed source software projects and three open source software projects, Paulson et al. [138] found that generally OSS has fewer defects than closed source software. By contrast, Stamelos et al. [135] offered a structural code analysis and suggest that the code quality of an OSS was lower than the quality implied by an industrial standard. It seems likely that these results vary greatly by project, suggesting the need for further research on antecedents of code quality.

4.5.2 Open source software implementation. Outside of the mainstream of most FLOSS research, some studies have examined to how OSS is being adopted and used in different contexts [150-158]. Dinkelacker et al. [159] describe activities at Hewlett Packard that aim to adapt the benefits of open source software for internal use, through the progressive introduction of open source practices. They began with “inner source,” the opening of all code behind the corporate firewall, then “controlled source” which restricts access to contracted partners and finally “open source,” where ‘the community’ in general is invited to participate. Chan [160] examined the practices that surround the emergence of free software legislation in Peru. In addition to the research that studies

OSS adoption outside OSS communities, Verma et al. [161] explored factors that influence FLOSS adoption and use within two different open source communities: one in the U.S. and one in India. They found that the degree of compatibility with users' mode of work, and ease of use are the two significant factors that influence FLOSS use in the U.S. open source community, but for the India group, compatibility is the only significant factor.

4.5.3 Evolution. The literature on FLOSS evolution has focused on two aspects: the evolution of the product, which is the software developed in this context; and the evolution of the community to that develops and maintains the software. Different types of FLOSS projects have different patterns of system evolution and community evolution [162].

Evolution of the software. Research confirms that the evolution of projects' size over time seems to contradict the laws of software evolution proposed for commercial software [163]. For example, Godfrey and Tu [164] observed that the evolution of the Linux Kernel did not obey Lehman's laws which say that "as the system grew, the rate of growth would slow, with the system growth approximating an inverse square curve".

Some literature looks in detail at code evolution patterns. Scacchi [64] notes that code tends to evolve incrementally rather than change radically. Capiluppi [165] found an unbalanced evolution patterns for some codes in an OSS project called ARLA – "some [code] branches may appear more appealing than others, and are extensively evolved, while other[s] remain in the same status for all the life cycle". Antoniol et al. [166] studied the duplication of code over time in the Linux kernel. They found that "Linux system does not contain a relevant fraction of code duplication. Furthermore, code duplication tends to remain stable across releases, thus suggesting a fairly stable structure, evolving smoothly without any evidence of degradation" (p.755).

Evolution of the community. Another focus is on the evolution of the community, which discusses the dynamic roles of developers and users over time. Oh and Jeon [167] discuss the impact of member retirement on community structure. They argued that a snowball effect might lead more members to leave when one member drops out, which might result in network separation and disintegration, so it may be important to maintain a balanced composition of all the different roles in a community [162], By studying three

FLOSS projects, Long and Siau [168] found that their interaction patterns evolve from a single hub at the beginning, to a core/periphery model as the projects mature.

Of course, code and community do not exist separately. They co-evolve and have an impact on each other. Nakakoji et al. [162] argues that the contribution made by members is the source of system evolution, and the system evolution in turn affects the contribution distribution among the developers, and thus redefines the roles of the contributors. Similarly, Capiluppi [165] argues that "when the tree structure reaches some status, the process of joining as a core developer seems to forestall" (p.23).

5 DISCUSSION

In the previous section we reviewed the empirical research on FLOSS development and use in an effort to assess the state of the literature. From the analysis we can see that we are still in the early stages of investigation of FLOSS and significant empirical work remains to understand this phenomenon.

The literature to date has been relatively limited in scope and many aspects of FLOSS development have received little examination. In this section, we discuss important areas that have remained under-researched and provide direction for future research. The analysis is again organized around the Input-Mediator-Output-Input (IMOI) model that was used in section 4. We also address a number of methodological and theoretical issues related to the study of FLOSS.

5.1 Inputs

There is an opportunity to pursue further research on FLOSS development inputs, particularly their impacts on the dependent variables. For example, sufficient detail has been provided regarding why individuals contribute to FLOSS development, but little work has been done to examine the impact of various motivations on individual behaviors in FLOSS development. It seems likely that motivations are linked to other facets of contribution, such as longevity of participation or achievement of leadership. For example, Hertel et al. [169] reports that future career prospects were significantly related to planned future activities, but not significantly related to actual contribution, suggesting that this motive might provide initial drive, but go unrealized. It would be an interesting finding to discover whether participants with particular types of

motivation are more likely to continue to contribute or to achieve leadership roles. Further, few studies have examined changes in motivation over time. Previous research has indicated that motivations may alter over time. For example, Ghosh [170] mentions that reputation was a stronger motivation amongst the longer term participants (who might be expected to actually have garnered reputation). But these analyses are preliminary and longitudinal analyses are needed to examine the phenomenon in detail. This is particularly important for insight on the rise of bounties (e.g., Gnome) and temporary corporate support of participants (e.g., Google's Summer of Code).

Software types and technologies used in FLOSS are two other interesting input variables that need further examination. Software of different types might attract different contributors and users, enable different coordination mechanisms, and establish different relationships with firms. Software types play an important role in FLOSS development, but surprisingly little work has examined the social implications of the differences in types of software produced by FLOSS developers. There is an opportunity to consider software type and its relationship to various aspects in FLOSS development in a more theoretically informed manner. Some questions that might provide interesting insights here are: How are software types related to individuals' participation in projects? How do software types impact firm involvement in FLOSS development? How do software types influence social processes such as decision making in FLOSS development?

Various tools (such as email lists, forums, bug track systems, CVS) play an important role in FLOSS development. In virtual team research, technologies used by team members are often examined to see how they coordinate team member activities, but few such studies have been done in the FLOSS context. Future research could further our understanding of which tools people actually use in FLOSS development, the influence that tools have upon the choice of a hosting site for a new project, the roles of different tools in FLOSS development, and how these tools interact and complement each other to support FLOSS development. Processes

Previous research on FLOSS development processes has focused on examining mechanisms used in different processes as described in Figure 5. More research is needed on factors that affect processes and how the characteristics of FLOSS development influence these processes. For example, few studies have touched on the impact of team diversity (e.g., team members' demographics, motivations, values, and skills) on their

collaboration. Further, how do external environmental factors, such as project type, company sponsored versus non-sponsored, interact with team and project development processes?

Social processes represent an area in which major gaps exist in the FLOSS research literature. In particular, little research has been conducted on social processes related to conflict management and team maintenance. Conflicts sometimes can have significant negative effects on FLOSS development, given its virtual and self-organizing nature. Team maintenance encompasses the pro-social, discretionary, and relation-building behaviors that create and maintain reciprocal trust and cooperation between members [171]. Theorists argue that team maintenance behavior is important because it is believed to be associated with team effectiveness. Several theories have been used to examine team maintenance in different contexts, such as social presence in text-based communication environments [172], face work in computer-mediated communications [173] and organizational citizenship behavior in traditional settings [174], but little research has been done on this topic in the FLOSS research literature and several important questions remain unanswered. What kind of factors likely trigger conflict? What's the role of leaders in conflict management? How team maintenance is created and sustained over time? Is there any relationship between project types and team maintenance behaviors?

Another potentially important factor is how projects manage the knowledge necessary for a successful development effort. Given its highly distributed environment and dynamic membership, FLOSS development faces particular knowledge management challenges. Previous research has explored various knowledge management activities such as knowledge creation and knowledge sharing. Additional research is needed in order to understand how members integrate knowledge from different sources. In particular, what mechanisms and team norms are used to store knowledge contributed by team members? What techniques are used to identify useful knowledge given the huge information flow?

Given the large number of in-depth studies of individual FLOSS projects, there are only a limited number of studies of projects that involve firm participation. This may be due to the relative difficulty of obtaining data from firms. But since one of the often cited reasons for studying FLOSS is the potential for adapting FLOSS practices to proprietary

production environments, additional research needs to be conducted to investigate how firm–involved FLOSS projects differentiate from non-firm-involved FLOSS projects. One particularly interesting topic might be how firm involvement in a FLOSS project changes project development over time.

5.2 Emergent States

To date, there has been little discussion of team members' interaction patterns over time. Roles are probably best studied as structurationally emergent in this context, but empirical FLOSS research has only touched on this [41]. Some other emergent states such as trust and shared mental models also remain understudied. Future research should further our understanding of how these emergent states form, maintain and change over time. Interesting outstanding questions include, What kinds of factors trigger these changes? Are there any patterns associated with different projects, and different project development phases? What is the relationship between processes and emergent states development? Outputs

Most current research on FLOSS team effectiveness uses objective measures such as downloads, code quality, bug fixing time, and number of developers. Behavioral measures, which are believed to impact members' desire to work together in the future, are typically missing. Since FLOSS development is usually a long-term project, it is important to include this measure in evaluating FLOSS effectiveness.

Another issue is that the link from output to input has not been addressed in previous literature. In a FLOSS developmental sequence, outputs become the inputs to future development. Although theorists have realized the importance of the cyclical causal feedback from outputs to inputs in team interactions [24], little empirical study incorporating this aspect of the phenomenon has been done in FLOSS research. More research is needed on how outputs contribute to or change inputs. For example, how do outputs such as user satisfaction impact team structure in the future?

5.3 Methodological and Theoretical issues

Finally, several methodological issues need to be addressed. First, a significant number of empirical studies of FLOSS have used archival data e.g., from SourceForge, and this type of data is not without problems. SourceForge provides only a limited amount of easily

コメント [AW6]: This c
to answer: yes, there are.
patterns associated with
characteristics? I'm not s
getting at in terms of diff

コメント [AW7]: or alv

available data which is both practically difficult and theoretically perilous to use in FLOSS research [175]. Archive data may also have a high omission rate [176], so more data sources are needed.

Second, a few studies use self-reported data, which can be problematic in terms of potential bias and accuracy. For example, some papers use self-reporting measures of hours spent in order to measure individual effort in FLOSS development. However, such data may be subject to reporting bias or demand effects, which may partly explain the lack of evidence for self-interested motivations. Or individuals may be driven by unexpressed, and therefore undocumented motivations. To overcome such potential biases, there is a need for such studies to incorporate objective measures of effort, such as CVS commits, patches applied, tracker involvement or even mailing list participation.

A third methodological concern with current FLOSS research is the sampling strategies used. For example, most research has studied well-established projects, not projects in the initial or transition phases. Many studies base their sampling on "top 1000" lists of popular projects on a particular project hosting site, introducing sampling biases that are rarely discussed or addressed. Samples that include projects with different hosts are very rare, and subject to concerns when quantitative data are used, as they may not be uniformly recorded. Most research has studied successful projects, not unsuccessful projects. Most research has studied only a few projects, usually less than 10 and often only one. There has also been insufficient attention to segmenting research by types of projects, e.g., based on the complexity of the project or the intended audience. Future research needs to compare projects in different phases of evolution and of varying types in order to advance our understanding of FLOSS development. Studies should also attempt to advance the frontier of research designs shown in Figure 3 by simultaneously studying larger samples of projects, in order to generalize the findings, and studying projects in more depth, with richer data.

Fourth, as with all studies of organizational phenomena, there is a strong need for careful attention to levels of data, analysis and theory. Multi-level studies raise several issues that need to be considered. Do the levels of aggregation used in theory and analysis match up appropriately? When individual level measures are used to evaluate group level phenomena, are the studies showing use of statistical tests to assure that aggregation to the group level is appropriate?

A final concern is with the paucity of longitudinal studies in FLOSS research. As the IMOI model suggests, team interactions are probably best studied over time, since their effects (such as coordination costs) should be felt only if the interactions actually occur in the same timeframe, which cross-sectional measures may not demonstrate. There is also little doubt that the FLOSS phenomenon has changed over the last 10 years, and continues to do so, for example with increasing corporate involvement, longitudinal research can detail the impact of such changes.

コメント **[AW8]:** This
– why would effects only
interaction was in the sa
think this is maybe rever
only demonstrated by cr
measures if their causes/
occurred in the same tim

6 CONCLUSIONS

Our goal in this article was to synthesize the empirical research on FLOSS development to date in order to clarify what we know and do not know. Of course, any attempt to capture a fast moving phenomenon is likely to suffer from some limitations, but the growing importance of the topic, reflected in the volume of research, makes it important to take stock of what has been done and to suggest promising directions for further work. To stimulate such work, we have presented a set of research questions around the Input-Mediators-Output-Input model, which provides a useful framework for further understanding the FLOSS phenomenon.

REFERENCES

[1] Free Software Foundation *The Free Software Definition*. City, 2006.
[2] Perens, B. *The open source definition*. O'Reilly, City, 1999.
[3] Kelty, C. M. *Two Bits - The Cultural Significance of Free Software*. 2008.
[4] Lee, G. K. and Cole, R. E. From a firm-based to a community-based model of knowledge creation: The case of Linux kernel development. *Organization Science*, 14, 6 2003), 633–649.
[5] Fitzgerald, B. The transformation of Open Source Software. *MIS Quarterly*, 30, 4 2006).
[6] Lakhani, K. R. and Wolf, R. G. *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*. MIT Press, City, 2005.
[7] Henkel, J. Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy*, 352006), 953–969.
[8] O'Mahony, S. Guarding the commons: How community managed software projects protect their work. *Research Policy*, 32, 7 2003), 1179.
[9] Spinellis, D., Gousios, G., Karakoidas, V. and Louridas, P. Evaluating the Quality of Open Source Software. *Electronic Notes in Theoretical Computer Science*, 2332009), 5-28.
[10] Walli, S., Gynn, D. and Rotz, B. v. *The Growth of Open Source Software in Organizations. A report* 2005.
[11] Ghosh, R. A. *Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU*. UNU-Merit, 2006.

[12] Alho, K. and Sulonen, R. *Supporting virtual software projects on the Web*. City, 1998.
[13] von Hippel, E. Innovation by user communities: Learning from open-source software. *Sloan Management Review*, Summer 2001), 82–86.
[14] von Hippel, E. and von Krogh, G. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, 14, 2 2003), 209–213.
[15] Wayner, P. *Free For All*. HarperCollins, New York, 2000.
[16] Raymond, E. S. The Cathedral and the Bazaar. *First Monday*, 3, 3 1998).
[17] O'Leary, M. B. and Cummings, J. N. The Spatial, Temporal, and Configurational Characteritics of Geographic Dispersion in Teams. *MIS Quarterly*, 31, 3 2007), 433-452.
[18] Watson-Manheim, M. B., Chudoba, K. M. and Crowston, K. Discontinuities and Continuities: a new way to understand virtual work. *Information, Technology & People*, 15, 3 (2002 2002), 191-209.
[19] Colllins, J. and Drucker, P. *A Conversation between Jim Collins and Peter Drucker*. City, 1999.
[20] Scacchi, W. Understanding the requirements for developing Open Source Software systems. *IEE Proceedings Software*, 149, 1 2002), 24–39.
[21] Scacchi, W. *Free and open source software development: Recent research results and methods*. Elsevier Press, City, 2007.
[22] Rossi, M. A. *Decoding the "Free/Open Source (F/OSS) Software Puzzle": A survey of theoretical and empirical contributions*. Universiti degli Studi di Siena, Dipartimento Di Economia Politica, 2004.
[23] von Krogh, G. and von Hippel, E. The Promise of Research on Open Source Software. *Management Science*, 52, 7 2006), 975-983.
[24] Ilgen, D. R., Hollenbeck, J. R. and Johnson, M. Team in Organizations: From Input-Process-Output Models to IMOI models. *Annual Review of Psychology*, 562005), 517-543.
[25] McGrath, N. What exactly are the merits of open-source software? Microsoft and Novell go head to head to talk out the issues - View#1. *Iee Review*, 50, 10 (Oct 2004), 46-48.
[26] Hackman, J. R. and Morris, C. G. *Group tasks, group interaction process, and group performance effectiveness: A review and proposed integration*. Academic Press, City, 1978.
[27] Marks, M. A., Mathieu, J. E. and Zaccaro, S. J. A Temporally Based Framework and Taxonomy of Team Processes. *Academy of Management Review*, 26, 3 2001), 356-376.
[28] Martins, L. L., Gilson, L. L. and Maynard, M. T. Virtual Teams: What do We Know and Where do We Go from Here? *Journal of Management*, 30, 6 (2004 2004), 805-835.
[29] Powell, A., Piccoli, G. and Ives, B. Virtual Teams: a review of current literature and directions for future research. *The DATA BASE for Advances in Information Systems*, 35, 1 2004), 6-36.
[30] Lancashire, D. The fading altruism of Open Source development. *First Monday*, 6, 12 2001).
[31] Gonzalez-Barahona, J. M., Robles, G., Andradas-Izquierdo, R. and Ghosh, R. A. Geographic origin of libre software developers. *Information Economics and Policy*, 20, 4 2008), 356 - 363.
[32] Dempsey, B. J., Weiss, D., Jones, P. and Greenberg, J. Who is an open source software developer? *Communications of the Acm*, 45, 2 (Feb 2002), 67-72.
[33] Tuomi, I. Evolution of the Linux credits file: Methodological challenges and reference data for Open Source resea. *First Monday*, 9, 6 2004).
[34] Lerner, J. and Tirole, J. Some simple economics of open source. *Journal of Industrial Economics*, 50, 2 (Jun 2002), 197-234.
[35] Lerner, J. and Tirole, J. The economics of technology sharing: open source and beyond. *Journal of Economic Perspectives*, 19, 2 2005), 99-120.
[36] Hann, I.-H., Roberts, J. and Slaughter, S. A. *Why developers participate in open source software projects: An empirical investigation*. City, 2004.
[37] Hann, I.-H., Roberts, J., Slaughter, S. and Fielding, R. *Economic incentives for participating in open source software projects*. City, 2002.
[38] Hars, A. and Ou, S. S. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6, 3 (Spr 2002), 25-39.

[39] Orman, W. H. Giving It Away for Free? The Nature of Job-Market Signaling by Open-Source Software Developers. *Advances in Economic Analysis & Policy*, 8, 1 2008), 4.
[40] Ghosh, R. A. FM Interview with Linus Torvalds: What motivates free software developers? *First Monday*, 3, 3 1998).
[41] Ye, Y. and Kishida, K. *Toward an Understanding of the Motivation of Open Source Software Developers*. City, 2003.
[42] Shah, S. K. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52, 7 2006), 1000--1014.
[43] Lakhani, K. R. and von Hippel, E. How open source software works: "free" user-to-user assistance. *Research Policy*, 32, 6 (Jun 2003), 923-943.
[44] Fang, Y. and Neufeld, D. Understanding Sustained Participation in Open Source Software Projects. *Journal of Management Information Systems*, 25, 4 2009), 9-50.
[45] Wu, C.-G., Gerlach, J. H. and Young, C. E. An empirical analysis of open source software developers' motivations and continuance intentions. *Information & Management*, 44, 3 2007), 253--262.
[46] Freeman, S. The material and social dynamics of motivation: Contributions to Open Source language technology development. *Science Studies*, 20, 2 2007), 55--77.
[47] Roberts, J., Hann, I.-H. and Slaughter, S. A. Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science*, 52, 7 2006), 984-999.
[48] David, P. A. and Shapiro, J. S. Community-based production of open-source software: What do we know about the developers who participate? *Information Economics and Policy*, 20, 4 2008), 364--398.
[49] Xu, B., Jones, D. R. and Shao, B. Volunteers' involvement in online community based software development. *Information & Management*, 46, 3 2009), 151 - 158.
[50] Subramanyam, R. and Xia, M. Free/Libre Open Source Software development in developing and developed countries: A conceptual framework with an exploratory study. *Decision Support Systems*, 46, 1 2008), 173.
[51] Rossi, C. and Bonaccorsi, A. *Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach?* 2005-02, opensource.mit.edu, 2005.
[52] West, J. and O'Mahony, S. *Contrasting Community Building in Sponsored and Community Founded Open Source Projects*. City, 2005.
[53] Bonaccorsi, A. and Rossi, C. Comparing Motivations of Individual Programmers and Firms to Take Part in the Open Source Movement. *Knowledge, Technology, and Policy*, 18, 4 2006), 40-64.
[54] Henkel, J. Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy*, 352006), 953-969.
[55] Luthiger Stoll, B. *Fun and Software Development*. 2005-07, opensource.mit.edu, 2005.
[56] Lakhani, K. R. and von Hippel, E. How open source software works: "free" user-to-user assistance. *Research Policy*, 32, 6 2003), 923-943.
[57] Crowston, K., Wei, K., Li, Q. and Howison, J. *Core and periphery in Free/Libre and Open Source software team communications*. City, 2006.
[58] Robles, G. and Gonzalez-Barahona, J. M. a. M. M. *Evolution of volunteer participation in libre software projects: Evidence from Debian*. City, 2005.
[59] Fershtman, C. and Gandal, N. Open source software: Motivation and restrictive licensing. *International Economics and Economic Policy*, 4, 2 2007), 209--225.
[60] O'Mahony, S. Guarding the commons: how community managed software projects protect their work. *Research Policy*, 32, 7 2003), 1179-1198.
[61] Stewart, K. J. and Gosain, S. *The Impact of Ideology on Effectiveness in Open Source Software Development Teams (updated on 08/2005*. 2005-08, opensource.mit.edu, 2005.
[62] Lerner, J. and Tirole, J. The Scope of Open Source Licensing. *Journal of Law Economics & Organization*, 21, 1 2005), 20-56.
[63] Colazo, J. A., Fang, Y. and Neufeld, D. J. *Development Success in Open Source Software Projects: Exploring the Impact of Copylefted Licenses*. City, 2005.
[64] Scacchi, W. Free/Open Source Software Development Practices in the Computer Game Community. *IEEE Software*, 21, 1 2004), 56-66.

[65] Michlmayr, M. *Managing Volunteer Activity in Free Software Projects*. 2004-07, opensource.mit.edu, 2004.
[66] Robbins, J. E. *Adopting OSS Methods by Adopting OSS Tools*. City, 2002.
[67] Lanzara, G. F. and Morner, M. l. *Making and sharing knowledge at electronic crossroads: the evolutionary ecology of open source*. City, 2004.
[68] Glance, D. G. Release criteria for the Linux kernel. *First Monday*, 9, 4 2004).
[69] Yamauchi, Y., Yokozawa, M., Shinohara, T. and Ishida, T. *Collaboration with lean media: How open-source software succeeds*. City, 2000.
[70] Mockus, A., Fielding, R. T. and Herbsleb, J. D. Two case studies of open source software development: Apache and Mozilla. *Acm Transactions on Software Engineering and Methodology*, 11, 3 (Jul 2002), 309-346.
[71] MacCormack, A., Rusnak, J. and Baldwin, C. Y. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. *Management Science*, 52, 7 2006), 1015-1030.
[72] Stark, J. *Peer reviews as a quality management technique in open-source software development projects*. City, 2002.
[73] Zimmermann, T., Weissgerber, P., Diehl, S. and Zeller, A. Mining Version Histories to Guide Software Changes. *IEEE Transactions on Software Engineering*, 31, 6 2005), 429.
[74] Dinh-Trong, T. T. and Bieman, J. M. The FreeBSD project: A replication case study of open source development. *Ieee Transactions on Software Engineering*, 31, 6 (Jun 2005), 481-494.
[75] Erenkrantz, J. R. *Release Management Within Open Source Projects*. City, 2003.
[76] Leibovitch, E. The business case for Linux. *IEEE Software*, 16, 1 1999), 40-44.
[77] Raisinghani, M. S. Search Engine Technology: A Closer Look at Its Future. *Information Resources Management Journal*, 18, 2 2005), I.
[78] von Krogh, G., Spaeth, S. and Lakhani, K. R. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32, 7 2003), 1217-1241.
[79] Ducheneaut, N. *The reproduction of Open Source Software programming communities*. 2003.
[80] Jensen, C. and Scacchi, W. *Collaboration, Leadership, Control, and Conflict Negotiation in the Netbeans.org Open Source Software Development Community*. City, 2005.
[81] Gacek, C. and Arief, B. The many meanings of Open Source. *IEEE Software*, 21, 1 2004), 34-40.
[82] Shaikh, M. and Cornford, T. *Version Management Tools: CVS to BK in the Linux Kernel*. City, 2003.
[83] Moon, J. Y. and Sproull, L. Essence of distributed work: The case of Linux kernel. *First Monday*, 5, 11 2000).
[84] German, D. M. The GNOME project: A case study of open source, global software development. *Software Process: Improvement and Practice*, 8, 4 2003), 201-215.
[85] Fielding, R. T. Shared leadership in the Apache project. *Communications of the ACM*, 42, 4 1999), 42–43.
[86] Giuri, P., Rullani, F. and Torrisi, S. Explaining leadership in virtual teams: The case of open source software. *Information Economics and Policy*, 20, 4 2008), 305 - 315.
[87] Fleming, L. and Waguespack, D. *Penguins, Camels, and Other Birds of a Feather: Brokerage, Boundary Spanning, and Leadership in Open Innovation Communities*. 2005-04, opensource.mit.edu, 2005.
[88] Scozzi, B., Crowston, K., Eseryel, U. Y. and Li, Q. *Shared Mental Models among Open Source Software*. City, 2008.
[89] Evans, P. and Wolf, B. Collaboration rules. *Harvard Business Review*, 83, 7 2005), 96-103.
[90] Sadowski, B. M., Sadowski-Rasters, G. and Duysters, G. Transition of governance in a mature open software source community: Evidence from the Debian case. *Information Economics and Policy*, 20, 4 2008), 323 - 332.
[91] Fielding, R. T. Shared leadership in the Apache Project. *Association for Computing Machinery. Communications of the ACM*, 42, 4 1999), 42.

[92] Mateos-Garcia, J. and Steinmueller, W. E. The institutions of open source software: Examining the Debian community. *Information Economics and Policy*, 20, 4 2008), 333 - 344.
[93] Malone, T. W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborn, C. S., Bernstein, A., Herman, G., Klein, M. and O'Donnell, E. Tools for inventing organizations: Toward a handbook or organizational processe. *Management Science*, 45, 3 (March 1999 1999), 425-443.
[94] van Wendel de Joode, R. and Egyedi, T. M. Handling variety: the tension between adaptability and interoperability of open source software. *Computer Standards & Interfaces*, 28, 1 2005), 109-121.
[95] Sagers, G. W. *The influence of network governance factors on success in open source software development projects*. City, 2004.
[96] Kuwabara, K. Linux: A bazaar at the edge of chaos. *First Monday*, 5, 3 2000).
[97] den Besten, M. L., Dalle, J.-M. and Galia, F. The allocation of collaborative efforts in open-source software. *Information Economics and Policy*, 20, 4 2008), 316--322.
[98] Koch, S. and Schneider, G. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, 12, 1 (Jan 2002), 27-42.
[99] Crowston, K. and Scozzi, B. *Coordination practices for bug fixing within FLOSS development teams*. City, 2004.
[100] Dafermos, G. *Management and Virtual Decentralized Networks: The Linux Project*. 2001-09, opensource.mit.edu, 2001.
[101] Asklund, U. and Bendix, L. *Configuration Management for Open Source Software*. City, 2001.
[102] Mockus, A., Fielding, R. T. and Herbsleb, J. D. *A case study of Open Source Software development: The Apache server*. City, 2000.
[103] Crowston, K., Wei, K., Li, Q., Eseryel, U. Y. and Howison, J. *Coordination of free/libre open source software development*. City, 2005.
[104] Crowston, K., Li, Q., Wei, K., Eseryel, U. Y. and Howison, J. Self-organization of teams in free/libre open source software development. *Information and Software Technology*, 492007), 564--575.
[105] Crowston, K. and Scozzi, B. Coordination practices within Free/Libre Open Source Software development teams: The bug fixing process. *Journal of Database Management*, 19, 2 2008), 1--30.
[106] van Wendel de Joode, R. Managing Conflicts in Open Source Communities. *Electronic Markets*, 14, 2 2004), 104-113.
[107] Becking, J., Course, S., Enk, G. v., Hangyi, H. T. and et al. MMBase: An open-source content management system. *IBM Systems Journal*, 44, 2 2005), 381.
[108] Ciborra, C. U. and Andreu, R. Sharing knowledge across boundaries. *Journal of Information Technology*, 16, 2 (Jun 2001), 73-81.
[109] Edwards, K. *Epistemic communities, situated learning and Open Source Software development*. City, 2001.
[110] Huysman, M. and Lin, Y. Learn to solve problems: a virtual ethnographic case study of learning in a GNU/Linux Users Group. *eJOV - The Electronic Journal for Virtual Organizations and Networks*, 7(December, 2005 2005), 56-69.
[111] Lee, G. K. and Cole, R. E. From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science*, 14, 6 (Nov-Dec 2003), 633-649.
[112] Hemetsberger, A. and Reinhardt, C. *Sharing and Creating Knowledge in Open-Source Communities: The case of KDE*. City, 2004.
[113] von Krogh, G., Spaeth, S. and Haefliger, S. *Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects*. City, 2005.
[114] Kuk, G. Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List. *Management Science*, 52, 7 2006), 1031-1042.
[115] Dahlander, L. Penguin in a new suit: a tale of how de novo entrants emerged to harness free and open source software communities. *Industrial and Corporate Change*, 16, 5 2007), 913-943.
[116] Bonaccorsi, A., Giannangeli, S. and Rossi, C. Entry Strategies Under Competing Standards: Hybrid Business Models in the Open Source Software Industry. *Management Science*, 52, 7 2006), 1085-1098.

[117] Dahlander, L. and Magnusson, M. G. Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy*, 34, 4 (May 2005), 481-493.
[118] Dahlander, L. and Magnusson, M. How do Firms Make Use of Open Source Communities? *Long Range Planning*, 41, 6 2008), 629 - 649.
[119] Stewart, K. J. and Gosain, S. *Impacts of ideology, trust, and communication on effectiveness in open source software development teams*. City, 2001.
[120] Gallivan, M. J. Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. *Information Systems Journal*, 11, 4 2001), 277-304.
[121] James, J. A preliminary study of the size determinant in small group interaction. *American Sociological Review*, 16, 4 1951), 474-477.
[122] Lin, Y. Hybrid Innovation: How Does the Collaboration Between the FLOSS Community and Corporations Happen? *Knowledge, Technology and Policy*, 18, 4 2006), 86-100.
[123] Crowston, K. and Howison, J. Hierarchy and centralization in Free and Open Source Software team communications. *Knowledge, Technology and Policy*, 18, 4 2005), 65-85.
[124] Crowston, K., Howison, J., Masango, C. and Eseryel, U. Y. *Face-to-face interactions in self-organizing distributed teams*. City, 2005.
[125] Cannon-Bowers, J. A. and Salas, E. *Shared mental models in expert decision making*. Lawrence Erlbaum Associates, City, 1993.
[126] Curtis, B., Walz, D. and Elam, J. J. *Studying the process of software design teams*. City, 1990.
[127] Scozzi, B., Crowston, K., Eseryel, U. Y. and Li, Q. *Shared mental models among open source software developers*. City, 2008.
[128] Crowston, K., Howison, J. and Annabi, H. Information systems success in Free and Open Source Software development: Theory and measures. *Software Process--Improvement and Practice*, 11, 2 2006), 123--148.
[129] Lee, S.-Y. T., Kim, H.-W. and Gupta, S. Measuring open source software success. *Omega*, 37, 2 2009), 426 - 438.
[130] Hecker, F. *Mozilla at one: A look back and ahead. Available at http://www.mozilla.org/mozilla-at-one.html*. City, 1999.
[131] Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z. and Offutt, A. J. Determining the Distribution of Maintenance Categories: Survey versus Measurement. *Empirical Software Engineering*, 8, 4 2003), 351-365.
[132] Samoladas, I., Stamelos, I., Angelis, L. and Oikonomou, A. Open source software development should strive for even greater code maintainability. *Communications of the Acm*, 47, 10 (Oct 2004), 83-87.
[133] Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z. and Offutt, A. J. *Maintainability of the Linux Kernel*. City, 2003.
[134] Bezroukov, N. A second look at the Cathedral and the Bazaar. *First Monday*, 4, 12 1999).
[135] Stamelos, I., Angelis, L., Oikonomou, A. and Bleris, G. L. Code quality analysis in open source software development. *Information Systems Journal*, 12, 1 (Jan 2002), 43-60.
[136] Gyimothy, T., Ferenc, R. and Siket, I. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Transactions on Software Engineering*, 31, 10 2005), 897+.
[137] Schmidt, D. P. Intellectual property battles in a technological global economy: A just war analysis. *Business Ethics Quarterly*, 14, 4 (Oct 2004), 679-693.
[138] Paulson, J. W., Succi, G. and Eberlein, A. An empirical study of open-source and closed-source software products. *Ieee Transactions on Software Engineering*, 30, 4 (Apr 2004), 246-256.
[139] Hanson, V. L., Brezin, J. P., Crayne, S., Keates, S. and et al. Improving Web accessibility through an enhanced open-source browser. *IBM Systems Journal*, 44, 3 2005), 573.
[140] Nichols, D. M., Thomson, K. and Yeates, S. A. *Usability and open-source software development*. City, 2001.

[141] van Wendel de Joode, R. and de Bruijne, M. *The Organization of Open Source Communities: Towards a Framework to Analyze the Relationship between Openness and Reliability*. City, 2006.
[142] Harris, J. S. Mission-critical development with open source software: Lessons learned. *Ieee Software*, 21, 1 (Jan-Feb 2004), 42.
[143] Subramaniam, C., Sen, R. and Nelson, M. L. Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, 46, 2 2009), 576--585.
[144] Stewart, K. J. and Ammeter, T. *An exploratory study of factors influencing the level of vitality and popularity of open source projects*. City, 2002.
[145] Méndez-Durón, R. and García, C. E. Returns from social capital in open source software networks. *J Evol Econ*, 192009), 277-295.
[146] Grewal, R., Lilien, G. L. and Mallapragada, G. Location, location, location: How network embeddedness affects project success in open source systems. *Management Science*, 52, 7 2006), 1043--1056.
[147] Long, J. and Yuan, M. J. *Are all Open Source Projects Created Equal? Understanding the Sustainability of Open Source Software Development Model*. City, 2005.
[148] Capra, E., Francalanci, C. and Merlo, F. An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects. *IEEE Transactions on Software Engineering*, 34, 6 2008), 765-782.
[149] Wynn, D. *Organizational Structure of Open Source Projects: A Life Cycle Approach*. City, 2003.
[150] Fitzgerald, B. and Kenny, T. *Open Source Software can Improve the Health of the Bank Balance - The Beaumont Hospital Experience*. 2003.
[151] Waring, T. and Maddocks, P. Open Source Software implementation in teh UK public sector: Evidence from the field and implications for the future. *International Journal of Information Management*, 25, 5 2005), 411.
[152] Yan, N., Leip, D. and Gupta, K. The use of open-source software in the IBM corporate portal. *IBM Systems Journal*, 44, 2 2005), 419.
[153] Fitzgerald, B. and Kenny, T. Developing an information systems infrastructure with open source software. *Ieee Software*, 21, 1 (Jan-Feb 2004), 50-+.
[154] Miralles, F., Sieber, S. and Valor, J. *CIO Herds and User Gangs in the Adoption of Open Source Software*. City, 2005.
[155] Goode, S. Something for nothing: management rejection of open source software in Australia's top firms. *Information & Management*, 42, 5 (Jul 2005), 669-681.
[156] Holck, J., Larsen, M. H. and Pedersen, M. K. *Managerial and technical barriers to the adoption of open source software*. City, 2005.
[157] Vemuri, V. K. and Bertone, V. Will the Open Source Movement Survive a Litigious Society? *Electronic Markets*, 14, 2 2004), 114.
[158] Bleek, W.-G. and Finck, M. *Migrating a Development Project to Open Source Software Development*. City, 2004.
[159] Dinkelacker, J., Garg, P. K., Miller, R. and Nelson, D. *Progressive Open Source*. ACM, City, 2002.
[160] Chan, A. Coding Free Software, Coding Free States: Free Software Legislation and the Politics of Code in Peru. *Anthropological Quarterly*, 77, 3 2004), 531-545.
[161] Verma, S., Jin, L. and Negi, A. *Open Source Adoption and Use: A Comparative Study Between Groups in the US and India*. City, 2005.
[162] Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. and Ye, Y. Evolution patterns of open-source software systems and communities2002), 85.
[163] Koch, S. Profiling an Open Source Project Ecology and Its Programmers. *Electronic Markets*, 14, 2 2004), 77-88.
[164] Godfrey, M. W. and Tu, Q. *Evolution in open source software: A case study*. City, 2000.
[165] Capiluppi, A. *Improving comprehension and cooperation through code structure*. City, 2004.
[166] Antoniol, G., Villano, U., Merlo, E. and Penta, M. D. Analyzing cloning evolution in the Linux kernel. *Information and Software Technology*, 44, 13 2002), 755.
[167] Oh, W. and Jeon, S. *Membership Dynamics and Network Stability in the Open-Source Community: The Ising Perspective*. City, 2004.

[168] Long, Y. and Siau, K. Social Network Structures in Open Source Software Development Teams. *Journal of Database Management*, 18, 2 2007), 25--40.
[169] Hertel, G., Niedner, S. and Herrmann, S. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32, 7 (Jul 2003), 1159-1177.
[170] Ghosh, R. A. *Free/Libre and Open Source Software: Survey and Study. Report of the FLOSS Workshop on Advancing the Research Agenda on Free / Open Source Software*. City, 2002.
[171] Ridley, M. *The Origins of Virtue: Human Instincts and the Evolution of Cooperation.* Viking, New York, 1996.
[172] Garrison, R., Anderson, T. and Archer, W. Critical thinking in a text-based environment: Computer conferencing in higher education. *The Internet and Higher Education*, 22000), 87-105.
[173] Morand, D. A. and Ocker, R. J. *Politeness theory and Computer-Mediated communication: A Sociolinguistic Approach to Analyzing Relational Messages.*, City, 2003.
[174] Morgan, E. L. The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. *Information Technology and Libraries*, 19, 2 2000), 105.
[175] Howison, J. and Crowston, K. *The perils and pitfalls of mining SourceForge*. City, 2004.
[176] Chen, K., Schach, S. R., Yu, L. G., Offutt, J. and Heller, G. Z. Open-source change logs. *Empirical Software Engineering*, 9, 3 2004), 197-210.