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

KEVIN CROWSTON, KANGNING WEI, JAMES HOWISON,
and ANDREA WIGGINS, Syracuse University

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 practices, social processes, and firm involvement practices), emergent states (e.g., social states 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 identify 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 and Programming teams*; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Sociology*; K.6.3 [**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

ACM Reference Format:
Crowston, K., Wei, K., Howison, J., and Wiggins, A. 2012. Free/Libre open-source software development: what we know and what we do not know. ACM Comput. Surv. 44, 2, Article 7 (February 2012), 35 pages.
DOI = 10.1145/2089125.2089127 http://doi.acm.org/10.1145/2089125.2089127

1. INTRODUCTION

In this article, we review the published empirical research literature on development of Free/Libre and Open-Source Software (FLOSS) development. FLOSS is an umbrella term covering a diversity of kinds of software and approaches to development, so we start this review by clarifying our focus. In general, the term free software[1] or open-source software refers to software released under a license that permits the inspection,

[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.

J. Howison is currently affiliated with the University of Texas at Austin.
K. Wei is currently affiliated with Shandong University, China.
Authors' addresses: K. Crowston and A. Wiggins, School of Information Studies, Syracuse University, Syracuse, NY 13244; emails: {crowston, awiggins}@syr.edu; K. Wei, School of Management, Shandong University, Jinan, Shandong, China; email: kwei@sdu.edu.cn; J. Howison, School of Information, University of Texas at Austin, Austin, TX 78701; email: jhowison@ischool.utexas.edu.

DOI 10.1145/2089125.2089127 http://doi.acm.org/10.1145/2089125.2089127

use, modification, and redistribution of the software's source code.[2] The distinction between free software and open-source software is sometimes controversial, and there are important differences between these two development communities [Kelty 2008]. However, our focus in this article is research on their development processes, which are acknowledged by participants to be largely similar (Free Software Foundation, http://www.fsf.org), hence our use of this umbrella term.

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 community-based development [Lee and Cole 2003], 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 [Fitzgerald 2006], but many projects exist outside of any formal organizational structure. Forges such as SourceForge and GForge are often used to organize FLOSS development efforts. 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 [Lakhani and Wolf 2005], even these contributions are often made available to the wider community [Henkel 2006]. As a result, FLOSS can be characterized as a privately produced public good [O'Mahony 2003]. 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, 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 [Vass 2007], 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 U.S. businesses use FLOSS [Walli et al. 2005]. Ghosh [2006] estimated the cost of recreating available FLOSS code at € 12B, noting that "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.

FLOSS also represents a different approach to innovation in the software industry. The research literature on software development and distributed work emphasizes the difficulties of distributed software development, but successful community-based FLOSS development presents an intriguing counterexample. 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

[2]See http://www.gnu.org/philosophy/free-sw.html for a definition and discussion of free software and http://www.opensource.org/docs/osd for a definition and discussion of open-source software.

of distributed work [Alho and Sulonen 1998]. 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 [von Hippel 2001, von Hippel and von Krogh 2003]. Teams are almost purely virtual, in that community-based developers contribute from around the world, meet face-to-face infrequently (if at all), and coordinate their activities primarily by means of computer-mediated communications (CMC) [Raymond 1998, Wayner 2000]. The teams have a high isolation index [O'Leary and Cummings 2007] 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 [Watson-Manheim et al. 2002], 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, which can be 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" [Collins and Drucker 1999]. As a result, research on FLOSS development offers lessons for many organizations.

However, as Scacchi [2002] 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 (e.g., Rossi [2006], Scacchi [2007], von Krogh and von Hippel [2006]) synthesize various major issues investigated in FLOSS research based on a small set of studies, 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 community-based FLOSS development and to suggest promising directions for further work.

This article 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. 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 article 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 in the following section.

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 not often 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[3] (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 addressing free software specifically 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. To ensure the quality of the review results, we retained only papers published in refereed journals and conference proceedings. Thus, working papers, unpublished dissertations, magazine articles, and opinion articles were excluded. 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 135 published empirical studies in which FLOSS development was the main theme.

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 later stage of collection and analysis. The quantitative analysis of the first wave of papers indicates that there

[3]Papers at `opensource.mit.edu` repository have been migrated to FLOSSHub (`http://flosshub.org`), which is a one-stop source for free/libre and open-source software research resources.

were no significant differences in the constructs studied between articles published in journals and conferences. Thus, the second wave of papers was collected only from journal articles published between 2006 and early 2009, yielding a further 49 empirical papers. The resulting 184 papers are from 52 different journals and 40 different conferences (Appendix 1 lists the journals and conferences[4]). The diversity of audiences for research on FLOSS development highlights the need for a systematic survey that pulls together work from these diverse sources, which might otherwise go unnoticed by researchers working in a single discipline.

One concern about this sampling approach is that in some research areas, particularly in computer science, research is presented primarily at conferences. Therefore, our coverage of the recent research (2006–9) may include the kinds of work done in disciplines where the major focus is on journal publication but overlook the kinds of work done in disciplines where only conference proceedings are more typical. However, we do note that we have a representation of computer science research in both phases of the study, including a number of computer science journal papers. On the other hand, including both conference and journal papers poses a possible threat of double counting papers, as in some disciplines (e.g., management), it is common for researchers to publish findings first in a conference paper, then as a revised version in a journal. Identifying when a paper should be counted as a revision of another paper can be difficult. Such duplication could again result in overrepresentation of certain kinds of research in the sample. We considered these possible sampling biases in designing the analysis approach, as we will describe.

We carried out two types of analysis that are presented in this article: a quantitative review and a qualitative review. The qualitative review, which is the main contribution of this article, includes papers from both waves of collection, while the quantitative review includes only papers 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 as noted above, making quantitative summaries of this sample suspect. The approach taken for these analyses is described in the following sections.

2.2. Quantitative Review Methodology

The goal of the quantitative review was to provide an overview of the nature of research being published on FLOSS development. For this review, each paper 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 or 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 (please refer to Appendix 2 for a presentation of the coding scheme[5]). 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 early 2006 and suggests gaps that future research might address.

[4]Appendix 1 is available at `http://floss.syr.edu/csur2012app1`.
[5]Appendix 2 is available at `http://floss.syr.edu/csur2012app2`.

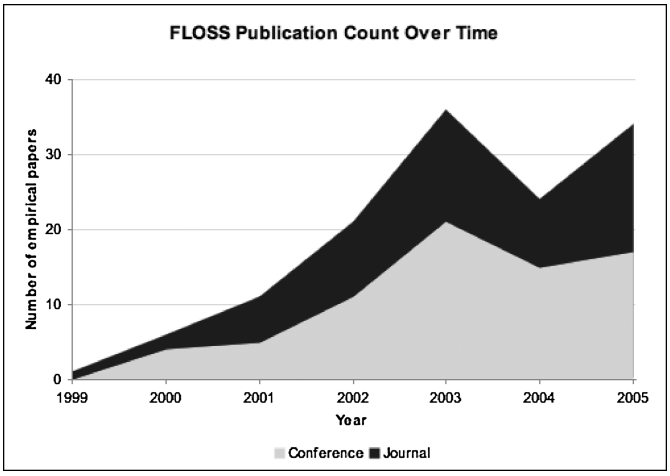

Fig. 1. Annual counts of empirical research publications.

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 our review 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 an inductive 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 transferred the codes onto sticky notes and sorted them as a group on a whiteboard. We then used the results of this sorting process to guide a search for relevant frameworks in the literature, leading to the selection of the model described below, which in turn was used to structure the review of the papers from both waves of paper collection. 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 next.

3. AN OVERVIEW OF THE FLOSS RESEARCH LITERATURE

In this section, 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, those published up through the early 2006. A little more than half of the sample (55%) were papers from conferences, with journal articles making up the remaining 45%. A sharp increase in the number of annual publications from 1999 through 2005 illustrated. 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 of our study, as noted previously.

3.1. Level of Analysis

Floss can be studied at different levels of analysis. We distinguished among studies at the artifact (which captures papers whose focus is on source code of FLOSS, technologies that support FLOSS development such as SourceForge, and programming or algorithms), 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 for research at the group or project level of analysis, which makes up 59% of the studies in the sample, with an additional 18% at the individual level, 19% at the organizational level, and just 4% at the societal level. About 7% of the studies focused on artifacts, with little discussion of behavioral aspects of FLOSS development.

3.2. Research Methods

Although a variety of research methods were observed, the case study was the most common, making up 43% of all papers in the sample (n = 58). Case studies were typically performed at the group level of analysis (n = 32, 24% of total), and based on archival 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 archival 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 archival data retrieved from development repositories. Only about 10% of papers used interview data, and likewise, only about 10% used data collected through observation. Multimethod 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%). Considering research methods and levels of analysis, the dominant form of FLOSS research overall was the study of a single project at the group level: 35 such papers comprised approximately 26% of our sample. This choice is closely related to the choices of research and data collection methods, as shown in Table I. To highlight the distribution of studies, the darkness of the cell and size of the font are proportional to the fraction of studies represented.

With respect to the projects studied in FLOSS literature, 42% of the papers sampled did not name the projects they studied, or did not study a specific 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 hundreds or thousands of projects). The remaining 58% of the sample provided some interesting insights into the types of open-source projects covered in the literature. The distribution of projects named in different studies showed a classic long-tail distribution, seen Figure 2. Linux was clearly the most commonly studied FLOSS project, appearing in 30 studies, followed by Apache (usually meaning the httpd server). Two-thirds of these papers studied only Linux whole the remaining third 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

Table I. Research Methods and Level of Analysis

Research Methods	Levels of Analysis						
	Total	Society	Organization	Group/Project	Individual	Artifact	Multi-level
Total		4%	19%	59%	18%	7%	8%
Case Study	43%	2%	8%	24%	9%	4%	4%
Survey	25%	1%	4%	13%	7%	1%	1%
Objects[1]	10%	1%	1%	7%	—	1%	—
Field Study	9%	1%	1%	6%	1%	—	—
Secondary Data[2]	4%	—	1%	4%	—	—	1%
Instrument Development[3]	4%	—	1%	2%	—	1%	1%
Multimethod	4%	1%	1%	2%	—	—	1%
Interview[4]	4%	1%	2%	2%	—	—	—
Simulation	2%	—	—	1%	1%	—	—
Experiment	1%	—	—	1%	—	—	—

Note: The following definitions were adapted from Alavi and Carlson [1992].

(1) As a research method, objects identify articles that describe a system, product, or project.
(2) Data used in the articles are collected by sources other than the researchers.
(3) Instrument development identifies papers that describe the development of instruments and/or measurements of FLOSS activities.
(4) As a research method, interview means the research is conducted by interviewing on an individual basis, which is different from using interviews as a data-collection technique.

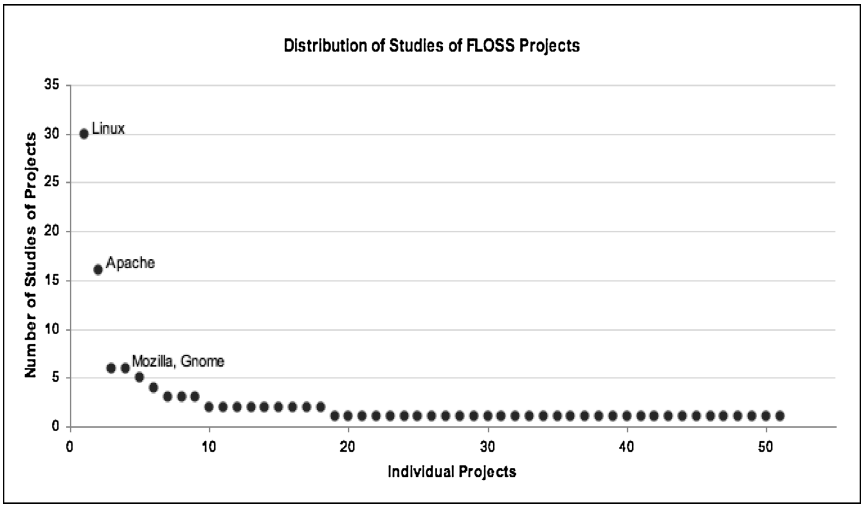

Fig. 2. Distribution of studies of FLOSS projects.

time. This suggests that as data on a wider variety of projects became more easily available, the variety of projects studied also rose.

As indicated by the distribution of projects studied in Figure 2, only 18 of the 51 projects (35%) named as subjects 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 trade-off 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 studies in that area. The figure shows the two types of studies that

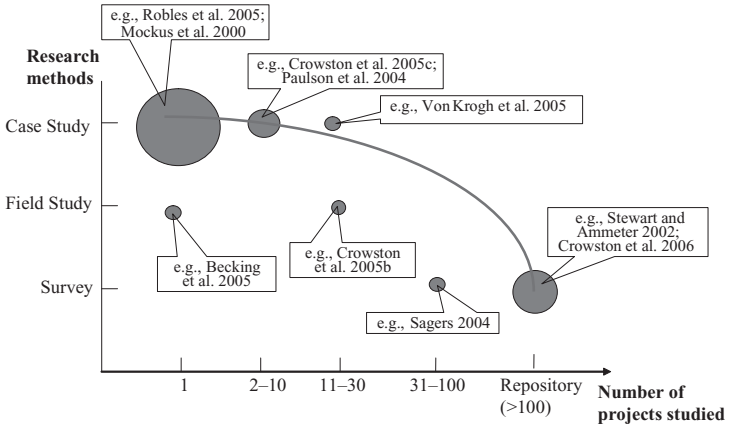

Fig. 3. Distribution of research by research methods and number of projects studied.

dominate current FLOSS research, just noted: (1) one or a small number of projects studied using the case study method or (2) survey 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 arc in Figure 3.

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. We identified the reference disciplines on which a paper was based by examining the theories or papers the authors used to formulate their models or hypotheses, and classified a paper as belonging to a particular reference discipline when it predominantly cited other papers from that discipline as the source [Vessey et al. 2002]. Approximately 20% of the papers did not explicitly refer to any reference disciplines, and about 64% referred to only a single reference discipline. The remaining 16% of the papers incorporated two to four reference disciplines, with business and management influences presented 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 comprised almost a third of the references as well, with information science and sociology being the next most common reference disciplines mentioned in the literature. While it may seem surprising that computer science and engineering are not comprise a larger fraction of the studies, the diversity of fields reflects our focus on empirical studies of development practices. In addition, 62% of papers that drew upon multiple reference disciplines employed theory in their studies, in contrast to only 27% of papers that referenced 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 theories. We classified a paper as "theory-included" when it explicitly cited existing theories/principles to support its models or hypotheses. Case studies and surveys made up the bulk of the papers in our sample, and were 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

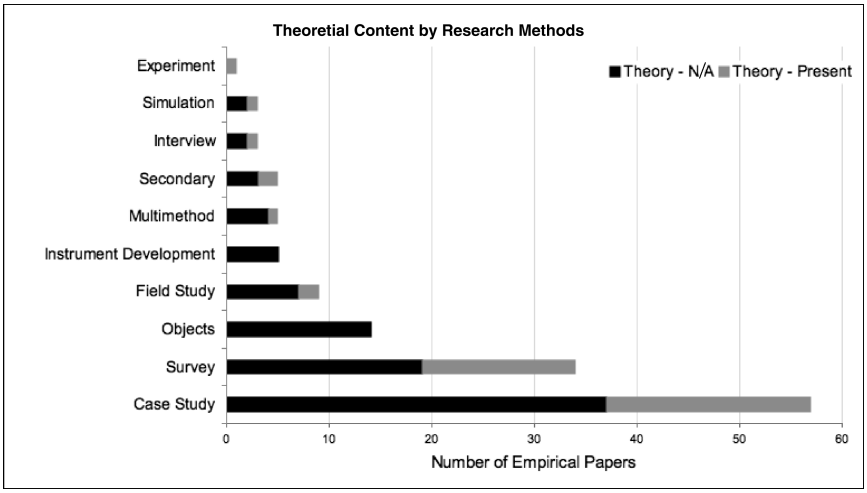

Fig. 4. Inclusion of references to theory by research methods.

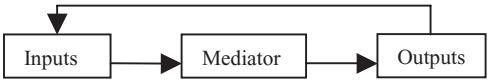

Fig. 5. Inputs-mediators-outputs-inputs model (adapted from Ilgen et al. [2005]).

or artifacts (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.

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 section, 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 sections 4.2–4.5, which forms the bulk of this article.

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 illustrated in Figure 5 [Ilgen et al. 2005], which draws together decades of work in the 'small group' literature [Hackman and Morris 1978; Marks et al. 2001; McGrath 1984, 1991, 1997]. This model most closely matched the inductive model and provided additional structure for the framework presented in this article. We chose the IMOI model over earlier input-process-output models (e.g., Hackman and Morris 1978) 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 [Ilgen et al. 2005]. The suitability of the

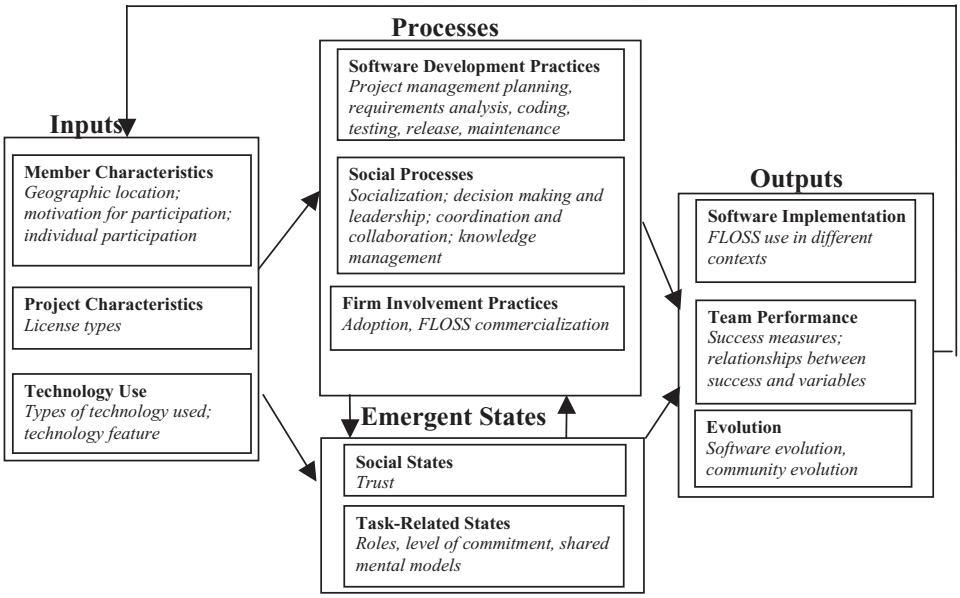

Fig. 6. Constructs studied in the reviewed FLOSS research papers and their relations.

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 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" [Marks et al. 2001]. Outputs represent task and non-task consequences of a team functioning [Martins et al. 2004].

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 was project characteristics, which made 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 were overwhelmingly studied through archival 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 archival 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 were most often used with the construct of social processes in the first wave, and instrument development was most frequently related to research methodology. The level of analysis was also relevant to the constructs studied. As we

have mentioned, the overwhelming majority of studies were at the group level. However, not all constructs conformed to this trend. Motivation was 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. (Please refer to Appendix 3 for a complete summary of coding for each paper we collected.[6])

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 [Martins et al. 2004; Powell et al. 2004]. Inputs that have been investigated by previous FLOSS research can be grouped under the labels of member characteristics, project characteristics, and technology use.

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

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 claimed that FLOSS involves global development efforts, dominated by European and North American participants, with a leading position of the former. For example, by examining the Linux CREDITS file on all major kernel releases and developer contact information on the GNOME project website, Lancashire [2001] 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. declines in influence (high absolute numbers but low relative numbers) while Europe, especially Northern Europe, grows in importance. Based on data from code repositories, Ghosh [2006] revealed Europe as the leading region in terms of globally active FLOSS software developers and global project leaders, followed closely by North America. Asia and Latin America face disadvantages at least partly due to language barriers, but may have an increasing share of developers active in local communities. 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. [2008] reported that North America and Europe (especially Germany, U.K. and France) are the top regions for FLOSS developers. Further more 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 [Dempsey et al. 2002; Tuomi 2004].

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., Lerner and Tirole [2002, 2005a]) who argued that motivation is 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

[6]Appendix 3 is available at `http://floss.syr.edu/csur2012app3`.

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 [Hann et al. 2004] and reward motives such as career development [Hann et al. 2002; Hars and Ou 2002; Orman 2008] are the two most frequently mentioned extrinsic motivations. Enjoyment-based motivations such as fun [Ghosh 1998] and sharing or learning opportunities [Shah 2006; Ye and Kishida 2003] are the two most commonly mentioned intrinsic motivations. User needs [Lakhani and von Hippel 2003; Lerner and Tirole 2002] are 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., Fang and Neufeld 2009; Shah 2006, and Wu et al. 2007]). 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 [2006] 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 [2007] 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 [2009] revealed that initial motivations to participate do not effectively predict long-term participation, but situated learning and identity construction behaviors are 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 individuals' participation in particular projects. (e.g., David and Shapiro 2008 and Roberts et al. 2006). For example, by studying 135 projects on SourceForge, Xu et al. [2009] 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 [2008] found that developers in different regions with similar project preferences are driven by different motivations. For instance, for projects that are larger in scale, more modular, and universal in nature, Chinese developers are found to be 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 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 [West and O'Mahony 2005]. Surveys have shown that as many as 45% of contributors are paid by firms for their participation, either directly or indirectly [Hars and Ou 2002]. Research on this topic has also examined the reasons that companies are investing internal resources in FLOSS development. For example, Bonaccorsi and Rossi [2006] 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 at the bottom of the list. In comparison with

individuals, they found that firms focus less on social motivations such as reputation and learning benefits. Similarly, by studying the firm-developed innovations embedded within Linux, Henkel [2006] emphasized the importance of receiving outside technical support as a motivator for revealing code.

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 [2005] found that participants spend an average of 12.15 hours per week, with project leaders averaging 14.13 hours bug-fixers and otherwise active users at around 5 hours per week. Lakhani and von Hippel [2003] studied the amount of time participants spend reading and answering user support questions in the Apache forums, finding that the most frequent answer providers spend 1.5 hours per week, but that frequent information seekers spend 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. [2006b] studied 120 relatively successful SourceForge projects, and found that the most common length of participation, across all roles, is no longer than a single month, reflecting a highly skewed distribution of participation. The tenure of participants varies significantly according to their roles. For example, Robles and Gonzalez-Barahona [2005] 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. Roberts et al. [2006] studied the impact of different motivations on individual contribution levels in Apache project. The results showed that 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 [2007] 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 used by a project that allow its intellectual property to be publicly and freely accessible and yet governable [O'Mahony 2003].

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 [Rossi 2006]. The definitions of different licenses are complex legal terms and significant ambiguities still remain about their interpretation [Lerner and Tirole 2005b]. In empirical research, licenses are usually described as falling into three classes according to their relative restrictiveness[7] [Fershtman and Gandal 2007; Lerner and Tirole 2005b]: unrestrictive (e.g., the Berkeley Software Definition (BSD) license), restrictive (e.g., Lesser General Public License (LGPL)), and highly restrictive (e.g., General Public License (GPL)).

[7]The label "restrictiveness" is used in the literature to refer to whether the modified versions of the software are also required to be open and whether the modified versions of the software may only be combined with software distributed under licenses that share the first requirement [Stewart et al. 2006; Lerner and Tirole, 2005b].

A few empirical studies have taken this framework to examine the influence of license choices on various aspects of FLOSS development. By examining the SourceForge projects, Lerner and Tirole [2005b] examined the relationships between project types and the license choices. For example, they found that highly restrictive licenses are more common for projects geared towards end-users, but they are significantly less common for projects aimed at software developers. Some research has examined the impact of license choices on FLOSS team effectiveness, but the results are mixed. Using data gathered from the Freshmeat website (now free (code): `http://freecode.com/`), Stewart et al. [2005, 2006] found that OSS projects that use a nonrestrictive license become 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. [2005] found a reverse result, indicating that copyleft/restrictive licenses are 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 [2004], who discussed the importance of software version control systems such as CVS or Subversion, 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 discussed the interrelation of CVS use and email use (i.e., developers checking code into the CVS repository discuss the patches via email). Michlmayr [2003] illustrated the importance of bug trackers to coordinate among those working on questions and answers.

A small body of research studied the tools developers or users use to share and exchange knowledge. For example, Robbins [2002] discussed nine types of commonly used OSS engineering tools and illustrated their impact on the software development process. Using data from the developer mailing lists of two open-source software projects, Lanzara and Morner [2004] argued 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 [Scacchi 2004]. 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 [Scacchi 2004]. In the following sections, we consider the research on more specific practices, using the systems development lifecycle [Blanchard and Fabrycky 2006] as an organizing structure.

Planning. It is commonly held that FLOSS projects do not engage in formal planning. For example, Glance [2004] examined the kernel change logs to determine the criteria applied for releasing the Linux kernel. She argued that a release contains whatever has 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 [Yamauchi et al. 2000]. Planning seems to be one contribution made by firms involved with projects [Fitzgerald 2006].

Software Requirements Analysis. Similar to the planning stage, FLOSS projects are often said to not conduct formal software requirements analyses. Scacchi [2004] stated 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. [2002] stated 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.

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 [2004] noted 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. [2006] reported that a redesign of Mozilla resulted in an architecture that is much more modular than its predecessors, which emphasized 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 [2004] noted that the Linux kernel release process is 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. Stark [2002] noted that peer review has been used in conventional development as well, and cited research showing that it works without meetings. Although this survey showed 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 noted 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 [2003] described several testing processes, such as the Linux Test Project and the Linux Stabilization Project. Dinh-Trong and Bieman [2005] noted that the FreeBSD project also has a more defined testing process.

Release. The nature of open code is that users can almost always access the latest version of the code. However, as the code is being worked on, it may or may not be stable, so many users prefer to use a "released" version of known quality. However, it is difficult to identify a single FLOSS approach to releases. Erenkrantz [2003] 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 [1998] to "release early; release often." For example, Glance [2004] found that in Linux, releases comes at an irregular rate. It is not clear what drives the schedule, but a release pattern is observed in terms of accepting patches for a while, then freezing acceptance of new code to allow the system to stabilize, though stability is assessed only by an absence of reported problems. In contrast, some projects have more organized approaches to releases. Dinh-Trong and Bieman [2005] reported that FreeBSD releases a new version every four months. A "Release Engineering Team" coordinates the release, following a pattern similar to that of Linux.

Maintenance. Maintenance is a primary activity in FLOSS development, as in conventional development. In FLOSS development, however, the nature of maintenance has been described as more like reinvention, which acts as "a continually emerging source of adaptation, learning, and improvement in FLOSS functionality and quality" [Scacchi 2004]. Studies of maintenance in FLOSS has focused on activities such as problem-solving processes, user-support practices [Lakhani and von Hippel 2003], 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 may be provided commercially [Leibovitch 1999]. Other projects have commercial sponsors who sell support (e.g., MySQL, SugarCRM). Smaller projects tend to rely on community support, for example, via email or discussion boards. However, Singh et al. [2006] analyzed help interactions and found that this process is often inefficient because initial posts lack the necessary information to answer questions, resulting in back-and-forth postings. The authors suggested that some details be captured automatically as part of initial reports, and also articulated the potential benefit of developing practices and tools for more explicitly reusing information, for example, 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 [Marks et al. 2001]. 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 nonparticipant 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. [2003] proposed 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. Duchenaut [2005] studied socialization in the Python project from both learning and political perspectives, and, confirming the findings of von Krogh et al. [2003], found that participants who move to the center of a project act in a way that expose 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 "enrolled allies" for their evolution towards the core of the project. He also highlighted the manner in which the onus for socialization falls almost entirely on the would-be developer, rather than the team. The process thus acts as a filter for participants that match the project. Because of the importance of attracting new developers, some projects have devoted efforts to making the project more accessible, though the success of these efforts does not appear to have been studied.

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 [Jensen and Scacchi 2005].

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 [2004] 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 [2003] 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 [2000] also pointed out that in Linux, Linus Torvalds originally made most of the key decisions for the team. German [2003] provided 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 [Fielding 1999]. 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 [Fitzgerald 2006].

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 [Giuri et al. 2008; Lerner and Tirole 2002]. 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 contributions [Scozzi et al. 2008], diversified skills [Giuri et al. 2008], and a structural position in their teams [Evans and Wolf 2005; Scozzi et al. 2008]. Second, FLOSS teams usually exhibit shared leadership instead of having a single leader [Sadowski et al. 2008]. According to Fielding [1999], 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-Garcia and Steinmueller [2008] 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 [Malone et al. 1999]. 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 [van Wendel de Joode and Egyedi 2005]. Coordination activities play an important role in FLOSS development, and are critical to project success [Sagers 2004] and to sustaining collective efforts in FLOSS teams, especially for large project such as Linux [Kuwabara 2000].

The backgrounds and characteristics of the different projects may influence the use of coordination mechanisms in general. For example, Java tends to use ex-ante mechanisms (i.e., coordination before taking action) while Linux tends to use 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 [van Wendel de Joode and Egyedi 2005; Yamauchi et al. 2000]. Using ten large FLOSS projects, den Besten et al. [2008] found that collaboration effort is 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 outputs [Crowston and Scozzi 2004, Dinh-Trong and Bieman 2005; Koch and Schneider 2002; Mockus et al. 2002]. Based on a theoretical framework of network governance, Sagers [2004] demonstrated 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 [Jensen and Scacchi 2005; Mockus et al. 2002]. An example is Linux, as Dafermos [2001] stated, "Modularity makes Linux an extremely flexible system and propels massive development parallelism and decreases the total need for coordination." However, Mockus et al. [2002] noted that while developers tend to work on the same modules repeatedly, most modules are 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 [2001].

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 community-based FLOSS development [Crowston et al. 2007; Crowston and Scozzi 2008; Crowston et al. 2005b; Mockus et al. 2000; Mockus et al. 2002].

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 [Jensen and Scacchi 2005, van Wendel de Joode and Egyedi 2005].

In addition to these coordination mechanisms, teams need mechanisms to manage conflict. From interviews, van Wendel de Joode [2004] 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 FLOSS development faces knowledge management (KM) challenges because of its highly distributed, knowledge-intensive characteristics [Becking et al. 2005; Ciborra and Andreu 2001; Edwards 2001]. This body of research focuses on how knowledge is shared or reused in FLOSS development [Dafermos 2001; Hemetsberger and Reinhardt 2004; Huysman and Lin 2005; Lakhani and von Hippel 2003; Lanzara and Morner 2004; Lee and Cole 2003; Singh et al. 2006; von Krogh et al. 2005]. For example, Huysman and Lin [2005] found 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 [2004] illustrated 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. [2005] reported on the reuse of knowledge in software development based on 15 open-source projects. The authors found 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 [2006] reported strategic interaction including conversational interactivity, cross-thread connectivity, and participation inequality expands knowledge sharing, but extreme concentration of participation would exert a negative effect on knowledge sharing.

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 [2003] 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 [2004] took 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.

4.3.3. 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 [Bonaccorsi et al. 2006; Dahlander 2007]. Research has revealed that firms usually adopt a hybrid production model by combining proprietary software and open-source software models [Bonaccorsi et al. 2006]. The strategies that firms use to create new or utilize existing FLOSS communities are also discussed Dahlander and Magnusson [2005]. For example, by studying four firms involved with FLOSS, Dahlander and Magnusson [2008] discovered three ways firms use to connect with FLOSS communities: (1) accessing development in the community in order to extend their resource base; (2) aligning their strategy with the work in the community and (3) 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 group research, and has been noted as a determining factor to achieving effective team collaboration. Researchers have also suggested that trust is important in FLOSS team development [Evans and Wolf 2005; Stewart and Gosain 2001]. Trust is often related to team effectiveness. For example, Stewart and Gosain [2001] proposed that shared ideology enables the development of affective and cognitive trust, which in turn leads to group efficacy and effectiveness. But not all researchers share the same belief. In a study of published case studies of FLOSS projects, Gallivan [2001] 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, but in another context, a support-question asker [Ye and Kishida 2003]. Most research on roles focuses on the differences between distinct roles and how to define roles. Gacek

and Arief [2004] suggested distinguishing between passive and active users, between non-developers and developers, and 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. [2006b] studied the distribution of contributions to the bug-tracking systems for 120 Sourceforge teams using three different methods: first, the self-reporting of teams based on the list of developers on the Sourceforge site; 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 is not a good indicator of core membership (at least in bug fixing), and that the skew of contribution in the communications domain is substantially higher than in the code domain. The more reliable measures peg the core group size at a median of three (about 5% of participants in the project). The results are in line with early results in sociology on feasible group sizes, such as James [1951].

Lin [2006] described 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 [Mockus et al. 2000, Mockus et al. 2002], communication contribution [Crowston and Howison 2005] and support contribution [Lakhani and von Hippel 2003]. 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. [2002] observed that the top 15 contributors (out of 388 total) have 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 [2005], in a study of the FreeBSD project, found that the top 15 contributors (of 354) contribute only 57% of new source code and one needs 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 window, but still found that the core group for FreeBSD is larger than that of Apache's. Koch and Schneider [2002], studying the GNOME project, also found lower skew (top 15 contributing only 48%) but argued 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 [1999] related that all the core participants in Apache have other "real jobs," which usually involve using the software for which they are a contributor. Lakhani and von Hippel [2003] found that Apache participants spend about 30% of their work time on Web servers.

Luthiger Stoll [2005] surveyed developers about the balance between work time and spare time. The author found that over 60% of the time spent contributing is 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 falls as spare time continues to rise (decreasing returns). This finding is supported by Michlmayr [2004], who reported that participants understand others to have "real" jobs that justifiably interfere with their participation. In addition, Crowston et al. [2005a] reported 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 [Cannon-Bowers and Salas 1993]. Research on software development, in particular, has identified the importance of shared understanding in the area of software development. Curtis et al. [1990], noted that "a fundamental problem in building large systems is the development of a common understanding of the requirements and design across the project team." They went 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. Scozzi et al. [2008] analyzed mental models in a FLOSS project using cognitive mapping and process analysis. Specifically, the authors compared the mental models of four developers from the Apache Lucene Java project. Their analysis suggested 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 that the sharing is 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. [2006a] identified seven 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. [2009] 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

[Crowston et al. 2006a], 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 [Bezroukov 1999; Hecker 1999; Samoladas et al. 2004; Schach et al. 2002; Schach et al. 2003], product/software quality [Glance 2004; Gyimothy et al. 2005; Michlmayr 2003; Schmidt and Porter 2001; Stamelos et al. 2002], defect density [Mockus et al. 2000; Paulson et al. 2004], usability [Hanson et al. 2005; Nichols et al. 2001; Schmidt and Porter 2001], reliability [Gyimothy et al. 2005; Harrison 2001; Leibovitch 1999; van Wendel de Joode and de Bruijne 2006], and value of software output [Harrison 2001].

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 sizes, project types, project life cycles, sponsorships, ideology, and license types. For example, based on longitudinal data on FLOSS projects hosted on SourceForge, Subramaniam et al. [2009] found that restrictive licenses (as defined in Section 4.2.2) have an adverse impact on FLOSS success. By surveying projects hosted on SourceForge Website, Stewart and Gosain [2006] found that OSS community ideology impacts team effectiveness in terms of attraction and retention of developer input and the generation of project outputs. In another study of 240 open-source projects registered on Freshmeat, Stewart and Ammeter [2002] 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 [Mendez-Duron and García 2009], network embeddedness [Grewal et al. 2006] and leadership [Long and Yuan 2005]. For example, based on 75 FLOSS projects, Capra et al. [2008] reported a high degree of openness in governance practices leads to higher software quality.

A few studies investigated 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 [2001] noted that although trust is rarely mentioned, ensuring control is an important criterion for effective performance within OSS projects. Wynn [2004] found the fit between the life cycle stage and the specific organizational characteristics of 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 compared 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. [2004] found that generally OSS has fewer defects than closed-source software. By contrast, Stamelos et al. [2002] offered a structural code analysis and suggested that the code quality of an OSS is lower than the quality implied by an industrial standard. It seems likely that these results vary greatly by projects, suggesting the need for further research on antecedents of code quality.

4.5.2. Open-Source Software Implementation. Outside of most mainstream FLOSS research, some studies have examined how OSS is being adopted and used in different contexts [Bleek and Finck 2004; Fitzgerald and Kenny 2003; Fitzgerald and Kenny 2004; Goode 2005; Holck et al. 2005; Miralles et al. 2005; Vemuri and Bertone 2004; Waring and Maddocks 2005; Yan et al. 2005]. Dinkelacker et al. [2002] described 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 was invited to participate. Chan [2004] 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. [2005] 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, which develops and maintains the software. Different types of FLOSS projects have different patterns of system evolution and community evolution [Nakakoji et al. 2002].

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 [Koch 2004]. For example, Godfrey and Tu [2000] observed that the evolution of the Linux Kernel does not obey Lehman's laws, which states 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 [2004] noted that code tends to evolve incrementally rather than change radically. Capiluppi [2004] found 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. [2002] 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 [2004] discussed 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 [Nakakoji et al. 2002]. By studying three FLOSS projects, Long and Siau [2007] found that project 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. [2002] argued 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, thus redefined the roles of the contributors. Similarly, Capiluppi [2004] argued that "when the tree structure reaches some status, the process of joining as a core developer seems to forestall" (p. 23).

5. DISCUSSION

In Section 4 we reviewed the empirical research on FLOSS development 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 investigating FLOSS and significant empirical work remains in order 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. [2003] reported that future career prospects are 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, although previous research has indicated that motivations may alter over time. For example, Ghosh [2002] mentioned that reputation is 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 types play an important role in FLOSS development and 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. For example, Stewart et al. [2006] found that organizational sponsorship influences the impact of licensing on development activities. However, limited 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. For example, we found that the results of research on the impact of license types on team effectiveness are mixed. One possible explanation might be that software types (e.g., based on the intended audience, software topics, environment, etc.) have mediating impacts, so it would be important to study how software types influence the impact of licenses on team effectiveness. Other questions that might provide interesting insights in this area: 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 members' 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.

5.2. Processes

Previous research on FLOSS development practices has pointed out that FLOSS development does not seem to adopt formal procedures in systems-development life cycle. But it might not be the case for some FLOSS projects in practice, especially major projects. Fitzgerald [2006] argued that "the open source phenomenon has undergone a significant transformation from its free software origins to a more mainstream, commercially viable form—OSS 2.0" (p. 587), and the development process becomes less bazaar-like as strategic planning becomes paramount. For example, many, if not all, of the major FLOSS projects have planning and requirements analysis mechanisms. For instance, a lot of discussion about future Mozilla products comes from monthly meetings of Mozilla Labs (e.g., `http://labs.mozilla.com/2008/07/monthly-labs-meetup-july-2008/`). As companies have increasingly adopted open-source software, release scheduling and planning are also different now. Both commercial and the large noncommercial projects (e.g., OpenOffice.org and Mozilla Firefox) have created roadmaps and schedules. So more research is needed to investigate how the adoption of these formal software-development practices influence FLOSS development in a longrun.

Previous research on FLOSS development processes has focused on examining mechanisms used in different processes as described in Figure 6. 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 individual 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 [Ridley 1996]. 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 [Garrison et al. 2000], face work in computer-mediated communications [Morand and Ocker 2003], and organizational citizenship behavior in traditional settings [Organ et al. 2006], 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 is the role of leaders in conflict management? How is team maintenance 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?

Despite the increasing commercialization of FLOSS, there are not many studies of the details of firm participation in projects. This lacuna 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.

5.3. Emergent States

To date, there has been less discussion of team members' interaction patterns over time. Roles are probably best studied as structured and emergent in this context, but empirical FLOSS research has only touched on this [Ye and Kishida 2003]. 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? Do different project characteristics and project development phases lead to the emergence of different emergent state patterns, and how? What is the relationship between processes and emergent states development?

5.4. 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 longterm 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 [Ilgen et al. 2005], few empirical studies incorporating this aspect of the phenomenon have 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 structures in the future?

5.5. 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, for example, from SourceForge, and this type of data is not without problems. SourceForge and other forges provide only a limited amount of easily available data, which is both practically difficult and theoretically perilous to use in FLOSS research [Howison and Crowston 2004]. Archival data may also have a high omission rate [Chen et al. 2004], so more and richer 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 on a project 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 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 ten, and often only one. There has also been insufficient attention to segmenting research by types of projects, for example, 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. FLOSS data can be collected at the level of single contributions, individual contributors, entire projects, or even project clusters, and different theories will be applicable at different levels. Multi-level studies in particular 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 ensure that aggregation to the group level is appropriate? For example, using project level data (e.g., project downloads) to make inferences about individual project members (e.g., effectiveness of different kinds of contributors) poses significant validity concerns.

A final concern is with the paucity of longitudinal studies in FLOSS research. There is little doubt that the FLOSS phenomenon has changed over the last ten years, and continues to do so. So team interactions are probably best studied over time, as the IMOI model suggests. For example, with increasing corporate involvement, longitudinal research can detail the impact of such changes.

6. CONCLUSIONS

The rapid development of FLOSS as an alternative way for large software-systems development calls for a need to examine its socio-technical work practices and development processes [Scacchi 2007]. Our goal in this article is to synthesize the empirical research on FLOSS development, to date, in order to clarify what we know and do not know. The Input-Mediators-Output-Input, model in Figure 6 emphasizes the interaction cycles between inputs, mediators, and outputs of FLOSS development. 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. In Section 5, we discussed a number of future research directions according to the model we proposed. We hope that our discussion will inspire additional discussions for future FLOSS research.

Empirical research on FLOSS development is still in its early stage and shows tremendous promise for future research. In order to advance our understanding of FLOSS phenomenon, researchers need to draw on theoretical foundations that have been utilized in prior research on social interaction and software development, as well as other theoretical bases that are relevant to FLOSS phenomenon, to develop a more theoretically grounded understanding of FLOSS development. With these steps,

studies on FLOSS development have the potential to inform researchers and practitioners about how to understand, interpret, and effectively manage FLOSS development.

REFERENCES

ALHO, K. AND SULONEN, R. 1998. Supporting virtual software projects on the Web. *Proceedings of the the 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

ANTONIOL, G., VILLANO, U., MERLO, E. AND PENTA, M. D. 2002. Analyzing cloning evolution in the Linux kernel. *Info. Softw. Technol. 44*, 13, 755–765.

ASKLUND, U. AND BENDIX, L. 2001. Configuration management for open source software. In *Proceedings of the 1st Workshop on Open Source Software Engineering*.

BECKING, J., COURSE, S., ENK, G. V. AND HANGYI, H. T. 2005. MMBase: An open-source content management system. *IBM Syst. J. 44*, 2, 381–397.

BEZROUKOV, N. 1999. A second look at the Cathedral and the bazaar. *First Monday 4*, 12.

BLANCHARD, B. S. AND FABRYCKY, W. J. 2006. *Systems Engineering and Analysis*. Prentice Hall.

BLEEK, W. AND FINCK, M. 2004. Migrating a development project to open source software development. In *Proceedings of the ICSE 4th Workshop on Open Source Software Engineering*.

BONACCORSI, A., GIANNANGELI, S., AND ROSSI, C. 2006. Entry strategies under competing standards: Hybrid business models in the open source software industry. *Manage. Science 52*, 7, 1085–1098.

BONACCORSI, A. AND ROSSI, C. 2006. Comparing motivations of individual programmers and firms to take part in the open source movement. *Knowl. Technol. Policy 18*, 4, 40–64.

CANNON-BOWERS, J. A. AND SALAS, E. 1993. *Shared Mental Models in Expert Decision Making. Individual and Group Decision Making*. Lawrence Erlbaum Associates, Hillsdale, NJ, 221–246.

CAPILUPPI, A. 2004. Improving comprehension and cooperation through code structure. In *Proceedings of the ICSE 4th Workshop on Open Source Software Engineering*.

CAPRA, E., FRANCALANCI, C., AND MERLO, F. 2008. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *IEEE Trans. Softw. Engin. 34*, 6, 765–782.

CHAN, A. 2004. Coding free software, coding free states: Free software legislation and the politics of code in Peru. *Anthropol. Quart. 77*, 3, 531–545.

CHEN, K., SCHACH, S. R., YU, L. G., OFFUTT, J., AND HELLER, G. Z. 2004. Open-source change logs. *Empir. Softw. Engin. 9*, 3, 197–210.

CIBORRA, C. U. AND ANDREU, R. 2001. Sharing knowledge across boundaries. *J. Info. Technol. 16*, 2, 73–81.

COLAZO, J. A., FANG, Y., AND NEUFELD, D. J. 2005. Development success in open source software projects: Exploring the impact of copylefted licenses. In *Proceedings of the Americas Conference on Information Systems (AMCIS'05)*.

COLLINS, J. AND DRUCKER, P. 1999. A Conversation between Jims Collins and Peter Drucker. *Drucker Found. News 7*, 4–5.

CROWSTON, K. AND HOWISON, J. 2005. Hierarchy and centralization in free and open source software team communications. *Knowl. Technol. Policy 18*, 4, 65–85.

CROWSTON, K., HOWISON, J., AND ANNABI, H. 2006. Information systems success in free and open source software development: Theory and measures. *Softw. Process Improv. Practice 11*, 2, 123–148.

CROWSTON, K., HOWISON, J., MASANGO, C., AND ESERYEL, U. Y. 2005. Face-to-face interactions in self-organizing distributed teams. In *Proceedings of the Academy Of Management Conference*.

CROWSTON, K., LI, Q., WEI, K., ESERYEL, U. Y., AND HOWISON, J. 2007. Self-organization of teams in free/libre open source software development. *Info. Softw. Technol. 49*, 564–575.

CROWSTON, K. AND SCOZZI, B. 2004. Coordination practices for bug fixing within FLOSS development teams. In *Proceedings of the 1st International Workshop on Computer Supported Activity Coordination*.

CROWSTON, K. AND SCOZZI, B. 2008. Coordination practices within free/libre open source software development teams: The bug fixing process. *J. Datab. Manag. 19*, 2, 1–30.

CROWSTON, K., WEI, K., LI, Q., ESERYEL, U. Y., AND HOWISON, J. 2005. Coordination of free/libre open source software development. In *Proceedings of the International Conference on Information Systems*.

CROWSTON, K., WEI, K., LI, Q., AND HOWISON, J. 2006. Core and periphery in Free/Libre and Open Source software team communications. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS'39)*.

CURTIS, B., WALZ, D., AND ELAM, J. J. 1989. Studying the process of software design teams. In *Proceedings of the 5th International Software Process Workshop On Experience With Software Process Models*, 52–53.

DAFERMOS, G. N. 2001. Management and virtual decentralised networks: the Linux project. *First Monday 6*, 11.

DAHLANDER, L. 2007. Penguin in a new suit: A tale of how de novo entrants emerged to harness free and open source software communities. *Indus. Corp. Change 16*, 5, 913–943.

DAHLANDER, L. AND MAGNUSSON, M. 2008. How do firms make use of open source communities? *Long Range Plan. 41*, 6, 629–649.

DAHLANDER, L. AND MAGNUSSON, M. G. 2005. Relationships between open source software companies and communities: observations from Nordic firms. *Res. Policy 34*, 4, 481–493.

DAVID, P. A. AND SHAPIRO, J. S. 2008. Community-based production of open-source software: what do we know about the developers who participate? *Inform. Econ. Policy 20*, 4, 364–398.

DEMPSEY, B. J., WEISS, D., JONES, P., AND GREENBERG, J. 2002. Who is an open source software developer? *Comm. ACM 45*, 2, 67–72.

DEN BESTEN, M. L., DALLE, J.-M., AND GALIA, F. 2008. The allocation of collaborative efforts in open-source software. *Inform. Econ. Policy 20*, 4, 316–322.

DINH-TRONG, T. T. AND BIEMAN, J. M. 2005. The FreeBSD project: a replication case study of open source development. *IEEE Trans. Softw. Engin. 31*, 6, 481–494.

DINKELACKER, J., GARG, P. K., MILLER, R., AND NELSON, D. 2002. Progressive Open Source. In *Proceedings of the International Conference on Software Engineering.*

DUCHENEAUT, N. 2005. Socialization in an open source software community: a socio-technical analysis. *Comput. Supp. Cooperat. Work 14*, 4, 323–368.

EDWARDS, K. 2001. Epistemic communities, situated learning and open source software development. In *Proceedings of the Epistemic Cultures and the Practice of Interdisciplinarity Workshop*.

ERENKRANTZ, J. R. 2003. Release management within open source projects. In *Proceedings of the International Conference on Software Engineering 3rd Workshop on Open Source Software Engineering*.

EVANS, P. AND WOLF, B. 2005. Collaboration rules. *Harvard Bus. Rev. 83*, 7, 96–103.

FANG, Y. AND NEUFELD, D. 2009. Understanding sustained participation in open source software projects. *J. Manag. Inform. Syst. 25*, 4, 9–50.

FERSHTMAN, C. AND GANDAL, N. 2007. Open source software: motivation and restrictive licensing. *Intern. Econ. Econ. Policy 4*, 2, 209–225.

FIELDING, R. T. 1999. Shared leadership in the Apache project. *ACM 42*, 4, 42–43.

FITZGERALD, B. 2006. The transformation of open source software. *MIS Quarter. 30*, 4, 587–598.

FITZGERALD, B. AND KENNY, T. 2003. Open source software in the trenches: lessons from a large-scale OSS implementation. In *Proceedings of International Conference on Information Systems*.

FITZGERALD, B. AND KENNY, T. 2004. Developing an information systems infrastructure with open source software. *IEEE Softw. 21*, 1, 50–55.

FREEMAN, S. 2007. The material and social dynamics of motivation: contributions to open source language technology development. *Sci. Stud. 20*, 2, 55–77.

GACEK, C. AND ARIEF, B. 2004. The many meanings of Open Source. *IEEE Softw. 21*, 1, 34–40.

GALLIVAN, M. J. 2001. Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. *Inform. Syst. J. 11*, 4, 277–304.

GARRISON, R., ANDERSON, T., AND ARCHER, W. 2007. Critical thinking in a text-based environment: computer conferencing in higher education. *Intern. High. Edu. 2*, 87–105.

GERMAN, D. M. 2003. The GNOME project: a case study of open source, global software development. *Softw. Process Improv. Practice 8*, 4, 201–215.

GHOSH, R. A. 1998. FM Interview with Linus Torvalds: what motivates free software developers? *First Monday 3*, 3.

GHOSH, R. A. 2002. Free/libre and open source software: survey and study. In *Proceedings of the Report of the FLOSS Workshop on Advancing the Research Agenda on Free/Open Source Software*.

GHOSH, R. A. 2006. Economic impact of open source software on innovation and the competitiveness of the *Inform. Comm. Technol. (ICT)*.

GIURI, P., RULLANI, F., AND TORRISI, S. 2008. Explaining leadership in virtual teams: the case of open source software. *Inform. Econ. Policy 20*, 4, 305–315.

GLANCE, D. G. 2004. Release criteria for the Linux kernel. *First Monday 9*, 4.

GODFREY, M. W. AND TU, Q. 2000. Evolution in open source software: a case study. In *Proceedings of the International Conference on Software Maintenance*.

GONZALEZ-BARAHONA, J. M., ROBLES, G., ANDRADAS-IZQUIERDO, R., AND GHOSH, R. A. 2008. Geographic origin of libre software developers. *Inform. Econ. Policy 20*, 4, 356–363.

GOODE, S. 2005. Something for nothing: management rejection of open source software in Australia's top firms. *Inform. Manag. 42*, 5, 669–681.

GREWAL, R., LILIEN, G. L., AND MALLAPRAGADA, G. 2006. Location, location, location: how network embeddedness affects project success in open source systems. *Manag. Sci. 52*, 7, 1043–1056.

GYIMOTHY, T., FERENC, R., AND SIKET, I. 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transact. Softw. Engin. 31*, 10, 897–910.

HACKMAN, J. R. AND MORRIS, C. G. 1978. Group tasks, group interaction process, and group performance effectiveness: a review and proposed integration. Group Processes, volume 8 of Advances in Experimental Social Psychology. L. Berkowitz, Eds., New York, Academic Press, 45–99.

HANN, I.-H., ROBERTS, J., SLAUGHTER, S., AND FIELDING, R. 2002. Economic incentives for participating in open source software projects. In *Proceedings of the 23rd International Conference on Information Systems*. 365–372.

HANN, I.-H., ROBERTS, J., AND SLAUGHTER, S. A. 2004. Why developers participate in open source software projects: an empirical investigation. In *Proceedings of the 25th International Conference on Information Systems*.

HANSON, V. L., BREZIN, J. P., CRAYNE, S., AND KEATES, S. 2005. Improving web accessibility through an enhanced open-source browser. *IBM Syst. J. 44*, 3, 573–588.

HARRISON, W. 2001. Editorial: open source and empirical software engineering. *Empir. Softw. Engin. 6*, 3, 193–194.

HARS, A. AND OU, S. S. 2002. Working for free? motivations for participating in open-source projects. *Int. J. Electr. Commerce 6*, 3, 25–39.

HECKER, F. 1999. Mozilla at one: a look back and ahead. `http://www-archive.mozilla.org/mozilla-at-one.html`.

HEMETSBERGER, A. AND REINHARDT, C. 2004. Sharing and creating knowledge in open-source communities: the case of KDE. In *Proceedings of the the 5th European Conference on Organizational Knowledge, Learning, and Capabilities*.

HENKEL, J. 2006. Selective revealing in open innovation processes: the case of embedded Linux. *Res. Policy 35*, 953–969.

HERTEL, G., NIEDNER, S., AND HERRMANN, S. 2003. Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux kernel. *Res. Policy 32*, 7, 1159–1177.

HOLCK, J., LARSEN, M. H., AND PEDERSEN, M. K. 2005. Managerial and technical barriers to the adoption of open source software. In *Proceedings of the Cots-Based Software Systems*, 289–300.

HOWISON, J. AND CROWSTON, K. 2004. The perils and pitfalls of mining SourceForge. In *Proceedings of the 26th International Conference on Software Engineering Workshop on Mining Software Repositories*.

HUYSMAN, M. AND LIN, Y. 2005. Learn to solve problems: a virtual ethnographic case study of learning in a GNU/Linux users group. *Electron. J. Virtual Org. Netw. 7*, 56–69.

ILGEN, D. R., HOLLENBECK, J. R., AND JOHNSON, M. 2005. Team in Organizations: from input-process-output models to IMOI models. *Ann. Rev. Psych. 56*, 517–543.

JAMES, J. 1951. A preliminary study of the size determinant in small group interaction. *Amer. Sociol. Rev. 16*, 4, 474–477.

JENSEN, C. AND SCACCHI, W. 2005. Collaboration, leadership, control, and conflict negotiation and the netbeans.org open source software development community. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*.

KELTY, C. M. 2008. *Two Bits: the Cultural Significance of Free Software*. Duke University Press, Durham, NC.

KOCH, S. 2004. Profiling an open source project ecology and its programmers. *Elect. Markets 14*, 2, 77–88.

KOCH, S. AND SCHNEIDER, G. 2002. Effort, cooperation and coordination in an open source software project: GNOME. *Infor. Syst. J. 12*, 1, 27–42.

KUK, G. 2006. Strategic interaction and knowledge sharing in the KDE developer mailing list. *Manag. Sci. 52*, 7, 1031–1042.

KUWABARA, K. 2000. Linux: a bazaar at the edge of chaos. *First Monday 5*, 3.

LAKHANI, K. R. AND VON HIPPEL, E. 2003. How open source software works: "free" user-to-user assistance. *Res. Policy 32*, 6, 923–943.

LAKHANI, K. R. AND WOLF, R. G. 2005. Why hackers do what they do: understanding motivation and effort in free/open source software projects. In Perspectives on Free and Open Source Software. J. Feller, B. Fitzgerald, S. Hissam and K. R. Lakhani. Eds. MIT Press, Cambridge, MA.

LANCASHIRE, D. 2001. Code, culture and cash: the fading altruism of open source development. *First Monday 6*, 12.

LANZARA, G. F. AND MORNER, M. L. 2004. Making and sharing knowledge at electronic crossroads: the evolutionary ecology of open source. In *Proceedings of the 5th European Conference on Organizational Knowledge, Learning and Capabilities*.

LEE, G. K. AND COLE, R. E. 2003. From a firm-based to a community-based model of knowledge creation: the case of the Linux kernel development. *Org. Sci. 14*, 6, 633–649.

LEE, S.-Y. T., KIM, H.-W., AND GUPTA, S. 2009. Measuring open source software success. *Omega 37*, 2, 426–438.

LEIBOVITCH, E. 1999. The business case for Linux. *IEEE Softw. 16*, 1, 40–44.

LERNER, J. AND TIROLE, J. 2002. Some simple economics of open source. *J. Indust. Econ. 50*, 2, 197–234.

LERNER, J. AND TIROLE, J. 2005a. The economics of technology sharing: open source and beyond. *J. Econ. Perspec. 19*, 2, 99–120.

LERNER, J. AND TIROLE, J. 2005b. The scope of open source licensing. *J. Law Econ. Org. 21*, 1, 20–56.

LIN, Y. 2006. Hybrid innovation: How does the collaboration between the FLOSS community and corporations happen? *Knowledge, Tech. Policy 18*, 4, 86–100.

LONG, J. AND YUAN, M. J. 2005. Are all open source projects created equal? Understanding the sustainability of open source software development model. In *Proceedings of the Americas Conference on Information Systems*.

LONG, Y. AND SIAU, K. 2007. Social network structures in open source software development teams. *J. Datab. Manag. 18*, 2, 25–40.

LUTHIGER STOLL, B. 2005. Fun and software development. In *Proceedings of the 1st International Conference on Open Source Systems*. 273–278.

MACCORMACK, A., RUSNAK, J., AND BALDWIN, C. Y. 2006. Exploring the structure of complex software designs: an empirical study of open source and proprietary code. *Manag. Sci. 52*, 7, 1015–1030.

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. 1999. Tools for inventing organizations: toward a handbook or organizational processes. *Manag. Sci. 45*, 3, 425–443.

MARKS, M. A., MATHIEU, J. E., AND ZACCARO, S. J. 2001. A temporally based framework and taxonomy of team processes. *Acad. Manag. Rev. 26*, 3, 356–376.

MARTINS, L. L., GILSON, L. L., AND MAYNARD, M. T. 2004. Virtual teams: what do we know and where do we go from here? *J. Manag. 30*, 6, 805–835.

MATEOS-GARCIA, J. AND STEINMUELLER, W. E. 2008. The institutions of open source software: examining the Debian community. *Inf. Econ. Policy 20*, 4, 333–344.

MCGRATH, J. 1984. Groups: interaction and Performance, Prentice-Hall, Englewood Cliffs, NJ.

MCGRATH, J. E. 1991. Time, interaction, and performance (TIP): a theory of groups. *Sm. Group Res. 22*, 147–174.

MCGRATH, J. E. 1997. Small group research, that once and future field: An interpretation of the past with an eye to the future. *Group Dynam. Theory, Res. Prac. 1*, 1, 7–27.

MENDEZ-DURON, R. AND GARCÍA, C. E. 2009. Returns from social capital in open source software networks. *J. Evolution. Econ. 19*, 277–295.

MICHLMAYR, M. 2003. Quality and the reliance on individuals in free software projects. In *Proceedings of the International Conference on Software Engineering 3rd Workshop on Open Source Software Engineering*.

MICHLMAYR, M. 2004. Managing volunteer activity in free software projects. In *Proceedings of the USENIX Annual Technical Conference*. 93–102.

MIRALLES, F., SIEBER, S., AND VALOR, J. 2005. CIO herds and user gangs in the adoption of open source software. In *Proceedings of the European Conference on Information Systems*.

MOCKUS, A., FIELDING, R. T., AND HERBSLEB, J. D. 2000. A case study of open source software development: The Apache server. In *Proceedings of the International Conference on Software Engineering*.

MOCKUS, A., FIELDING, R. T., AND HERBSLEB, J. D. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Method. 11*, 3, 309–346.

MOON, J. Y. AND SPROULL, L. 2000. Essence of distributed work: the case of the Linux kernel. *First Monday 5,* 11.

MORAND, D. A. AND OCKER, R. J. 2003. Politeness theory and computer-mediated communication: a sociolinguistic approach to analyzing relational messages. In *Proceedings of the 36th Hawaii International Conference on System Sciences*.

NAKAKOJI, K., YAMAMOTO, Y., NISHINAKA, Y., KISHIDA, K., AND YE, Y. 2002. Evolution patterns of open-source software systems and communities. In *Proceedings of International Workshop on Principles of Software Evolution*. 76–85.

NICHOLS, D. M., THOMSON, K., AND YEATES, S. A. 2001. Usability and open-source software development. In *Proceedings of the Symposium on Computer Human Interaction.* 49–54.

O'LEARY, M. B. AND CUMMINGS, J. N. 2007. The spatial, temporal, and configurational characteritics of geographic dispersion in teams. *MIS Quart. 31*, 3, 433–452.

O'MAHONY, S. 2003. Guarding the commons: How community managed software projects protect their work. *Res. Policy 32*, 7, 1179–1198.

OH, W. AND JEON, S. 2004. Membership dynamics and network stability in the open-source community: the ising perspective. In *Proceedings of International Conference on Information Systems*.

ORGAN, D., PODSAKOFF, P., AND MACKENZIE, S. 2006. Organizational Citizenship Behavior: its Nature, Antecedents, and Consequences, SAGE Publications, Thousand Oaks, CA.

ORMAN, W. H. 2008. Giving it away for free? The nature of job-market signaling by open-source software developers. *B.E. J. Econ. Analy. Policy 8*, 1, Article 12.

PAULSON, J. W., SUCCI, G., AND EBERLEIN, A. 2004. An empirical study of open-source and closed-source software products. *IEEE Trans. Softw. Eng. 30*, 4, 246–256.

POWELL, A., PICCOLI, G., AND IVES, B. 2004. Virtual Teams: A review of current literature and directions for future research. *Datab. Adv. Inf. Syst. 35*, 1, 6–36.

RAYMOND, E. S. 1998. The cathedral and the bazaar. *First Monday 3*, 3.

RIDLEY, M. 1996. *The Origins of Virtue: Human Instincts and the Evolution of Cooperation*, Viking, New York.

ROBBINS, J. E. 2002. Adopting OSS methods by adopting OSS tools. In *Proceedings of the International Conference on Software Engineering 2nd Workshop on Open Source Software Engineering*.

ROBERTS, J., HANN, I.-H., AND SLAUGHTER, S. A. 2006. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Manag. Sci. 52*, 7, 984–999.

ROBLES, G. AND GONZALEZ-BARAHONA, J. M. A. M. M. 2005. Evolution of volunteer participation in libre software projects: evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*. 100–107.

ROSSI, M. A. 2006. Decoding the Free/open source (F/OSS) software puzzle: A survey of theoretical and empirical contributions. In *the Economics of Open Source Software Development.* J. Bitzer and P. J. H. Schroder. Elsevier Press, Amsterdam. 15–56.

SADOWSKI, B. M., SADOWSKI-RASTERS, G., AND DUYSTERS, G. 2008. Transition of governance in a mature open software source community: evidence from the Debian case. *Inf. Econ. Policy 20*, 4, 323–332.

SAGERS, G. W. 2004. The influence of network governance factors on success in open source software development projects. In *Proceedings of International Conference on Information Systems*.

SAMOLADAS, I., STAMELOS, I., ANGELIS, L., AND OIKONOMOU, A. 2004. Open source software development should strive for even greater code maintainability. *Comm. ACM 47*, 10, 83–87.

SCACCHI, W. 2002. Understanding the requirements for developing open source software systems. *IEE Proc. Softw. 149*, 1, 24–39.

SCACCHI, W. 2004. Free/open source software development practices in the computer game community. *IEEE Softw. 21*, 1, 56–66.

SCACCHI, W. 2007. Free/open source software development: Recent research results and methods. *Adv. Comput. 69*, 243–295.

SCHACH, S. R., JIN, B., AND WRIGHT, D. R. 2002. Maintainability of the Linux kernel. In *Proceedings of the 2nd Workshop on Open Source Software Engineering*.

SCHACH, S. R., JIN, B., WRIGHT, D. R., HELLER, G. Z., AND OFFUTT, A. J. 2003. Determining the distribution of maintenance categories: survey versus measurement. *Emp. Softw. Eng. 8*, 4, 351–365.

SCHMIDT, D. C. AND PORTER, A. 2001. Leveraging open-source communities to improve the quality and performance of open-source software. In *Proceedings of the International Conference on Software Engineeering 1st Workshop on Open Source Software Engineering*.

SCOZZI, B., CROWSTON, K., ESERYEL, U. Y., AND LI, Q. 2008. Shared mental models among open source software developers. In *Proceedings of the 41st Hawai'i International Conference on System Science.*

SHAH, S. K. 2006. Motivation, governance, and the viability of hybrid forms in open source software development. *Manag. Sci. 52*, 7, 1000–1014.

SHAIKH, M. AND CORNFORD, T. 2003. Version management tools: CVS to BK in the linux kernel. In *Proceedings of the ICSE 3rd Workshop on Open Source Software Engineering.*

SINGH, V., TWIDALE, M. B., AND RATHI, D. 2006. Open source technical support: A look at peer help-giving. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences.*

STAMELOS, I., ANGELIS, L., OIKONOMOU, A., AND BLERIS, G. L. 2002. Code quality analysis in open source software development. *Inf. Syst. J. 12*, 1, 43–60.

STARK, J. 2002. Peer reviews as a quality management technique in open-source software development projects. *Comp. Sci. 2349,* 340–350.

STEWART, K. J., AMMETER, A. P., AND MARUPING, L. M. 2005. A Preliminary analysis of the influences of licensing and organizational sponsorship on success in open source projects. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*.

STEWART, K. J., AMMETER, A. P., AND MARUPING, L. M. 2006. Impacts of license choice and organizational sponsorship on user interest and development activities in open source software projects. *Inf. Syst. Res. 17*, 2, 126–144.

STEWART, K. J. AND AMMETER, T. 2003. An exploratory study of factors influencing the level of vitality and popularity of open source projects. In *Proceedings of the 23rd International Conference on Information Systems*.

STEWART, K. J. AND GOSAIN, S. 2001. Impacts of ideology, trust, and communication on effectiveness in open source software development teams. In *Proceedings of the 22nd International Conference on Information Systems*.

STEWART, K. J. AND GOSAIN, S. 2006. The impact of ideology on effectiveness in open source software development teams. *MIS Quart. 30*, 2, 291–314.

SUBRAMANIAM, C., SEN, R., AND NELSON, M. L. 2009. Determinants of open source software project success: A longitudinal study. *Decis. Supp. Syst. 46*, 2, 576–585.

SUBRAMANYAM, R. AND XIA, M. 2008. Free/libre open source software development in developing and developed countries: a conceptual framework with an exploratory study. *Decis. Supp. Syst. 46*, 1, 173–186.

THOMAS, C. 2003. Improving verification, validation, and test of the linux kernel: the Linux stabilization project. In *Proceedings of the ICSE 3rd Workshop on Open Source Software Engineering*.

TUOMI, I. 2004. Evolution of the Linux credits file: Methodological challenges and reference data for open source research. *First Monday 9*, 6.

VAN WENDEL DE JOODE, R. 2004. Managing conflicts in open source communities. *Electron. Markets 14*, 2, 104–113.

VAN WENDEL DE JOODE, R., AND DE BRUIJNE, M. 2006. The organization of open source communities: towards a framework to analyze the relationship between openness and reliability. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*.

VAN WENDEL DE JOODE, R., AND EGYEDI, T. M. 2005. Handling variety: The tension between adaptability and interoperability of open source software. *Comp. Stand. Interf. 28*, 1, 109–121.

VASS, B. 2007. Migrating to open source: Have no fear. In *Proceedings of the 3rd DoD Open Conference: Deployment of Open Technologies and Architectures within Military Systems*.

VEMURI, V. K. AND BERTONE, V. 2004. Will the open source movement survive a litigious society? *Electr. Markets 14*, 2, 114–123.

VERMA, S., JIN, L., AND NEGI, A. 2005. Open source adoption and use: a comparative study between groups in the U.S. and India. In *Proceedings of the Americas Conference on Information Systems*.

VESSEY, I., RAMESH, V., AND GLASS, R. L. 2002. Research in Information Systems: an empirical study of diversity in the discipline and its journals. *J. Manag. Infor. Syst. 19*, 2, 129–174.

VON HIPPEL, E. 2001. Innovation by user communities: Learning from open-source software. *Mit Sloan Manag. Rev. 42*, 4, 82–86.

VON HIPPEL, E. AND VON KROGH, G. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. *Org. Sci. 14*, 2, 209–213.

VON KROGH, G., SPAETH, S., AND HAEFLIGER, S. 2005. Knowledge reuse in open source software: an exploratory study of 15 open source projects. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*.

VON KROGH, G., SPAETH, S., AND LAKHANI, K. R. 2003. Community, joining, and specialization in open source software innovation: a case study. *Res. Policy 32*, 7, 1217–1241.

VON KROGH, G. AND VON HIPPEL, E. 2006. The promise of research on open source software. *Manag. Sci. 52*, 7, 975–983.

WALLI, S., GYNN, D., AND ROTZ, B. V. 2005. The growth of open source software in organizations. http://dirkriehle.com/wp-content/uploads/2008/03/wp_optaros_oss_usage_in_organizations.pdf.

WARING, T. AND MADDOCKS, P. 2005. Open source software implementation in the U.K. public sector: evidence from the field and implications for the future. *Int. J. Inf. Manag. 25*, 5, 411–428.

WATSON-MANHEIM, M. B., CHUDOBA, K. M., AND CROWSTON, K. 2002. Discontinuities and continuities: a new way to understand virtual work. *Inf. Technol. People 15*, 3, 191–209.

WAYNER, P. 2000. *Free For All.* HarperCollins, New York.

WEST, J. AND O'MAHONY, S. 2005. Contrasting community building in sponsored and community founded open source projects. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*.

WU, C.-G., GERLACH, J. H., AND YOUNG, C. E. 2007. An empirical analysis of open source software developers' motivations and continuance intentions. *Inf. Manag. 44*, 3, 253–262.

WYNN, D. 2004. Organizational structure of open source projects: a life cycle approach. In *Proceedings of the 7th Annual Conference of the Southern Association for Information Systems.* 285–290.

XU, B., JONES, D. R., AND SHAO, B. 2009. Volunteers' involvement in online community based software development. *Inf. Manag. 46*, 3, 151–158.

YAMAUCHI, Y., YOKOZAWA, M., SHINOHARA, T., AND ISHIDA, T. 2000. Collaboration with lean media: how open-source software succeeds. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'00)*.

YAN, N., LEIP, D., AND GUPTA, K. 2005. The use of open-source software in the IBM corporate portal. *IBM Syst. J. 44*, 2, 419–425.

YE, Y. AND KISHIDA, K. 2003. Toward an understanding of the motivation of open source software developers. In *Proceedings of International Conference on Software Engineering (ICSE'03)*.

Received May 2008; revised July 2009, February 2010; accepted June 2010