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

AUTHOR 1
Affiliation 1
and
AUTHOR 2 AND AUTHOR 3
Affiliation 2

We review the empirical research on Free/Libre Open Source Software (FLOSS) development and use to assess the state of the literature.Oour review is organized around an input-mediator-output-input (IMOI) model. We start with a description of the articles selected for the review. We then discuss findings of this literature categorized into issues pertaining to inputs (e.g., member characteristics, technology use and project characteristics), processes (software development and social processes), emergent states (e.g., trust and task related states) and outputs (e.g. team performance, FLOSS implementation and project evolution). Based on this review, we suggest research questions, including methodological and theoretical issues, to guide future inquiry in this area.

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

1. INTRODUCTION

In this paper, we review the empirical literature on development of Free/libre Open Source Software (FLOSS), that is, software developed by distributed teams of programmers and released under a license allowing inspection, modification and redistribution of the software's source[1]. There are thousands of 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, OpenOffice), programming environments (e.g., Perl, Python, gcc) and even enterprise systems (e.g., eGroupware, Compiere, openCRX).

[1] FLOSS software is generally available without charge (captured in a phrase commonly used in the community: "free as in free beer"). Much (though not all) of this software is also "free software", meaning that derivative works must be made available under the same unrestrictive license terms as the original (captured as "free as in free speech", thus "libre"). We have chosen to use the acronym FLOSS rather than the more common OSS to acknowledge this dual meaning.

Over the past ten years, FLOSS has moved from an academic curiosity to the computing mainstream, with a concurrent increase in the amount of research examining this phenomenon. A review of this literature is important and timely for several reasons. First and foremost, FLOSS has become an important phenomenon to understand for its own sake. FLOSS is now a major social movement involving an estimated 800,000 programmers around the world [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 US businesses use FLOSS [Walli et al. 2005]. These systems have become an integral part of the infrastructure of modern society, making it critical to understand more fully how they are developed.

As well, FLOSS represents a new approach to innovation in the software industry. The research literature on software development and on distributed work emphasizes the difficulties of distributed software development, but the apparent success of FLOSS development presents an intriguing counter-example. Characterized by a globally distributed developer force and a rapid, reliable software development process, effective FLOSS development teams somehow profit from the advantages and overcome the challenges of distributed work [Alho and Sulonen 1998]. Ghosh [2006] estimates the cost of recreating the available FLOSS code at €12B, and notes "This code base has been doubling every 18-24 months over the past eight years, and this growth is projected to continue for several more years". 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 in which they are empowered and encouraged to participate.

In addition to its intrinsic merits, FLOSS has attracted great interest because it provides an accessible example of other phenomena of growing interest. For example, many researchers have turned to 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]. These teams are almost purely virtual teams in that developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily by means of computer-mediated communications (CMC) [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 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.

Another important feature of the FLOSS development process is that many developers contribute to projects as volunteers, usually without being paid at all; others are paid by their employers, but not directly by the project. As a result, recruiting and retaining new contributors is a critical success factor for a FLOSS project. These features make FLOSS teams extreme examples of self-organizing distributed teams, but they are not inconsistent with what many organizations are facing in recruiting and motivating professionals and in 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" [Colllins 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". Though as our review shows, a great number of studies have been conducted on FLOSS, there have been few efforts made to integrate these findings into a coherent body of knowledge. The few surveys done [Rossi 2004; Scacchi 2007] synthesize various major issues investigated in FLOSS research based on a small set of studies. Not explained is how their review processes informed these issues or their sample strategies. Indeed, it is increasingly clear that the term FLOSS includes groups with a wide diversity of practices, with varying degrees of effectiveness, but the dimensions of this space are still unclear. A key goal of our review is to synthesize the empirical work done to date in order to clarify what we know and do not know about FLOSS development and to suggest promising directions for further work.

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

2. METHODOLOGY

This section describes our effort to identify and classify relevant work as a basis for a systematic review. Our literature review 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. Since FLOSS research is relatively new, we decided to collect as many articles on FLOSS as possible, before refining the collection as described below.

The majority of papers included in the review were collected in early 2006 using three methods to search for appropriate literature. First, we collected all papers from the opensource.mit.edu working paper repository, journal special issues on FLOSS, FLOSS tracks in conferences such as Academy of Management and Association of Information Systems conferences, the International Conference on Open Source Software (organized by International Federation for Information Processing Working Group 2.13) conferences and International Conference on Software Engineering workshops. Second, we conducted a web search in ABI/Inform and Web of Science using "open source software" as the keyword. Finally, we also 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. As well, we added a few papers to the collection that we found as we wrote the review. However, given our purpose to review the findings of FLOSS research to clarify what we know and what we do not know about FLOSS, we decided to limit our review to published empirical studies where FLOSS development and use were the main themes. The resulting 138 papers are from 30 different journals and 41 different conferences.

Each article was then reviewed and coded on numerous dimensions: publication year, publication venue type (conference or journal), the level of analysis (e.g., group or individual), research methods (e.g., survey, case study), data collection methods, the number and names of projects studied, reference disciplines that support the research, theories applied (if any) and the main constructs examined. Specific categories for each dimension were developed by a group of coders until basic agreement was achieved on a sample group of papers. The full collection was then split between two coders working through a system which 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 allows us to assess the current state of FLOSS research as well as identify gaps and directions for future research as described, in the following sections.

3. ANALYSIS AND FINDINGS

In this section we present our analysis and findings. We first briefly present an overview of the collected articles in terms of the characteristics mentioned previously: level of analysis, research methods, data collection methods, reference disciplines, theories applied, and projects studied. We then provide a description of Inputs-Mediators-Outputs-Inputs (IMOI) model which is used to organize the constructs studied in the literature. A detailed review of the constructs follows, which forms the bulk of this paper.

3.1 An overview of the literature

As noted above, papers included in the review were coded on type of publication. A little more than half of the sample, 55%, were papers from conferences; journal papers make up the remaining 45%. A sharp increase in the number of annual publication from 1999 through 2005 (Figure 1) demonstrates the increasing interest in the topic.

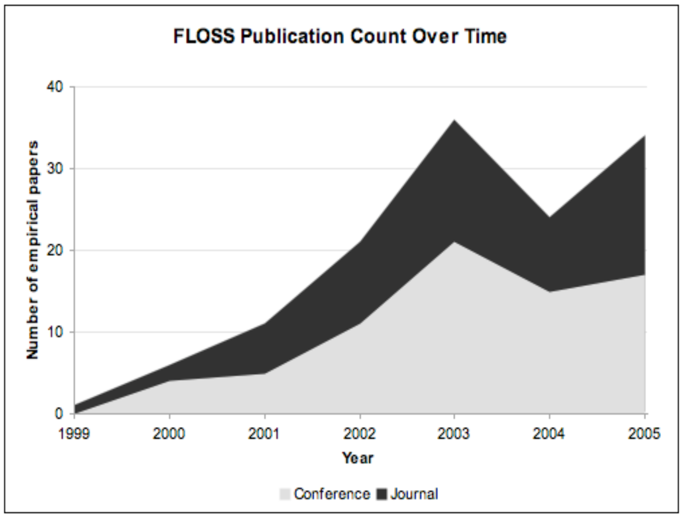

Figure 1: Annual counts of empirical research publications

3.1.1 Levels, Methods, Data and Theory. FLOSS can be studied at different levels of analysis. We distinguished among studies at the individual, group, organization and societal levels. Approximately 8% of papers included multiple levels of analysis, most often integrating the group and organization levels. The literature demonstrates a strong preference for the group (i.e., project) level of analysis, which makes up 57% of the studies in the sample, with an additional 17% at the individual level, 15% at the organization level, and just 4% at the level of society. The distribution of sample sizes of projects studied in these papers is also 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 is the study of a single project at the group level: the 35 such papers comprise approximately 26% of our sample. This choice is closely related to the choices of research and data collection methods, seen in Table 1.

Table 1. Research methods and level of analysis

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

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

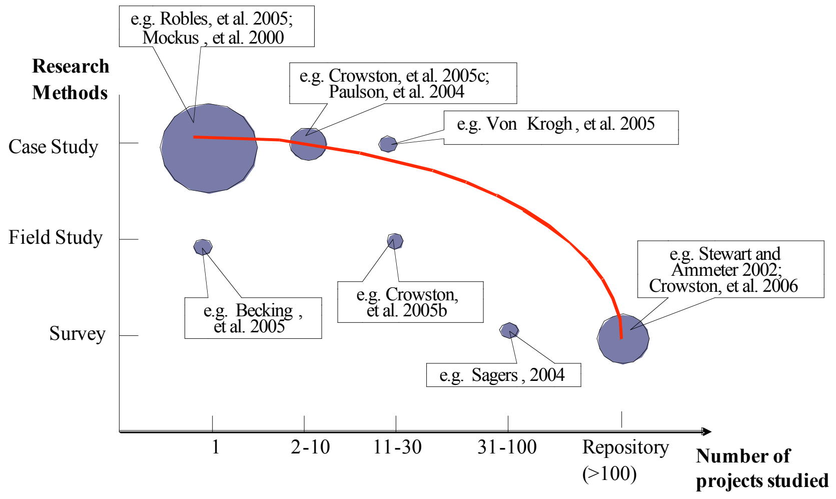

Figure 2. Distribution of Research by Research Methods and Number of Projects Studied

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

Figure 2 shows the tradeoff in FLOSS research between the sample sizes of projects studied and the intensity of the research approach. The size of the circle represents the relative number of the studies in that area. The figure shows the two types of studies that dominate current FLOSS research, as noted above: one or a small number of projects studied using the case study method or surveys covering a few variables to investigate larger sample sizes. These two types of studies represent two extremes of the tradeoff between the depth and breadth of a research study, anchoring the ends of a frontier of feasible research designs indicated by the red arc in Figure 2.

We also examined how studies used existing theory. Case studies and surveys make up the bulk of the papers in our sample, and are also the most likely to contain references to theory, as seen in Figure 3: 35% of case studies mentioned theoretical content, as did about 44% of surveys. While only about 32% of the overall sample contained references to theory, the more technical research approaches of instrument development and studies of objects (usually code) had no instances of theory usage. This demonstrates one of the challenges in describing an interdisciplinary body of research literature, as not all studies' contributions can be adequately judged based on the traditional classifications that we have applied here.

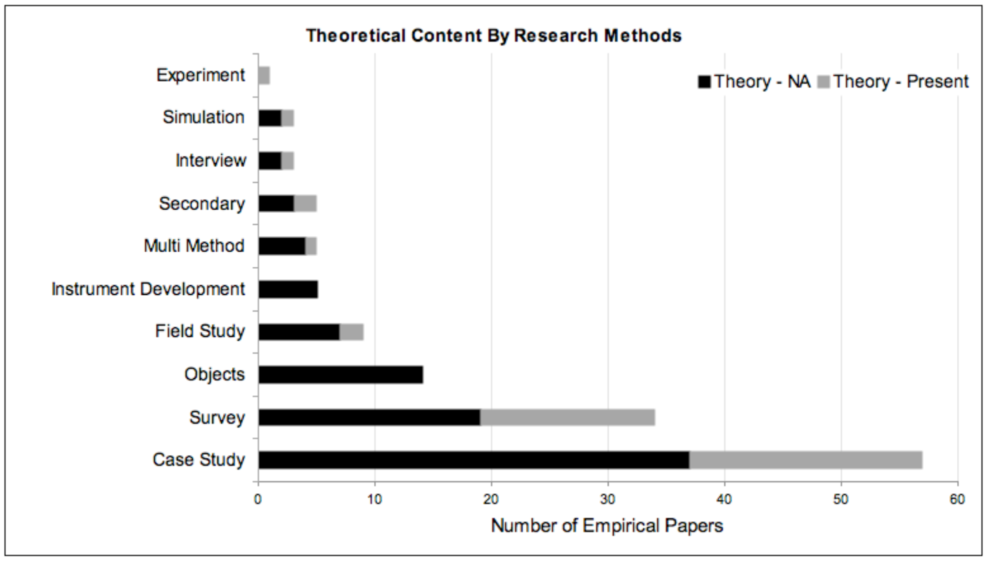

Figure 3. Theoretical Content by Research Methods

3.1.2 Projects. With respect to the projects that are studied in the FLOSS literature, 42% of the papers sampled did not name the projects they studied, or did not study an open source project, but rather studied some other aspects of FLOSS, such as implementation. None of the studies using repository data named the projects that were studied, presumably due to the scale of the research (these studies can include 100s or 1000s of projects). The remaining 58% of the sample provide some interesting insights into the types of open source projects studied in the literature. The distribution of projects named in different studies shows a classic long-tail distribution, seen in Figure 4. Linux is clearly the most commonly studied FLOSS project, appearing in 30 studies, followed by Apache. Two-thirds of these papers studied only Linux and the remaining third of the papers included additional projects besides Linux. However, while Linux and Apache have been most studied overall, the frequency of papers including these two projects peaked in 2003 and dropped sharply thereafter. At the same time, the number of studies with no projects named, often those examining a large sample of projects or using repository data (these factors correlate at r=0.98), have been increasing over time. This suggests that as data on a wider variety of projects became more easily available, the variety of projects studied also rose.

As indicated by the distribution of projects studied, shown in Figure 4, 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 representative the projects currently studied in FLOSS research are of 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.

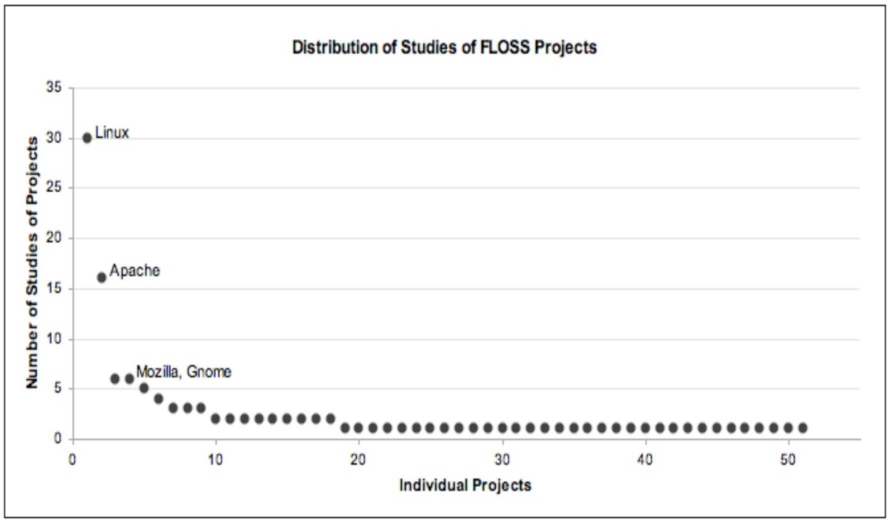

Figure 4. Distribution of Studies of FLOSS Projects

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

3.2 An organizing framework for the review

Having discussed the characteristics of the papers reviewed, we now turn to a review of their findings. A crucial task for a review paper is to provide a framework capable of organizing the existing literature and assisting future researchers in positioning their work in reference to that existing literature. We began our search for such a framework with a ground-up card sorting exercise. Four coders examined a sample of the literature and inductively recorded codes for the concepts studied in the paper. We then transferred these codes onto post-it notes and inductively sorted them on a white-board. We recorded the results and used them to guide us in a search for relevant frameworks in the literature. This search identified the inputs-mediators-outputs-inputs (IMOI) model (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 2004]. This model most closely matched the inductive model and provided additional structure for the framework presented in this paper. The suitability of the model is unsurprising since most FLOSS development does occur in small teams and the majority of the studies conducted research at the project level of analysis. Where necessary, we adapted the model to incorporate detailed constructs directly tied to the software engineering context of FLOSS work.

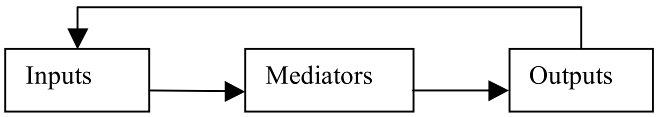

Figure 5. Inputs-Mediators-Outputs-Inputs Model (adapted from Ilgen et al. 2005)

Inputs represent starting conditions of a team, such as its member characteristics and project/task characteristics. Mediators represent variables that have mediational 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, p.357]. Outputs represent task and non-task consequences of a team functioning [Martins et al. 2004]. The reasons we chose the IMOI model over early Input-Process-Output models [e.g., Hackman and Morris 1978] to organize the articles are: 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]. Figure 6 presents the major concepts that we identified in current FLOSS research in each of these categories.

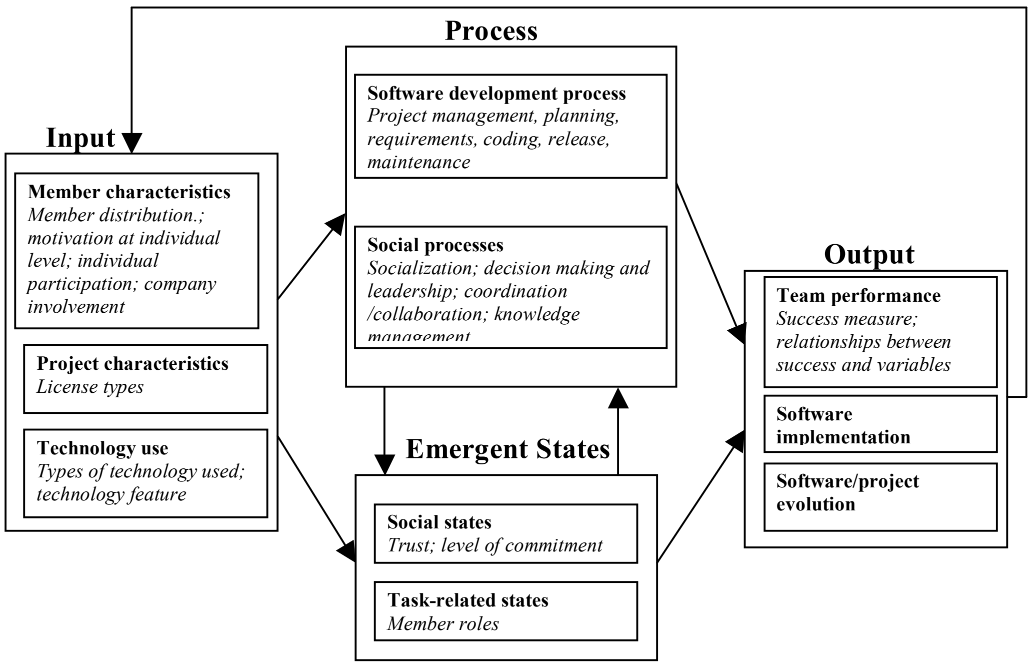
Process
Software development process
Project management, planning, requirements, coding, release, maintenance
Social processes
Socialization; decision making and leadership; coordination /collaboration; knowledge management
Input
Member characteristics
Member distribution.; motivation at individual level; individual participation; company involvement
Project characteristics
License types
Technology use
Types of technology used; technology feature
Emergent States
Social states
Trust; level of commitment
Task-related states
Member roles
Output
Team performance
Success measure; relationships between success and variables
Software implementation
Software/project evolution

Figure 6. Focus of Current FLOSS Research

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, indicating the descriptive nature of much of the FLOSS literature in our sample. Project characteristics were overwhelmingly studied through secondary data (15% of total), while constructs such as implementation and context did not rely as heavily on a single data type. Social processes (16%) and success (12%) were the next most frequent constructs observed, and studies of these constructs were also strongly reliant on secondary data. In contrast, studies of motivation tended to use questionnaire data more often than other types of data.

Certain research methods were also more strongly aligned with certain constructs; for example, field studies were most often used with the construct of social processes, 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 conform 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 review the literature, using this framework to structure our discussion.

3.3 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, technology use and context.

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

Geographic Location. There have been a number of studies of the geographical location of FLOSS participants. These studies have used self-reported descriptions of developer activity containing geographical information that are by-products of project practices. For example, Lancashire [2001] examined the Linux CREDITS file on all major kernel releases and developer contact information on the GNOME project website to explain the international distribution of FLOSS developers. The data on raw numbers of contributors showed a predominance of developers in the United States. Northern Europe is also well represented, but other regions, such as the Middle East, Africa, Asia and South America, are underrepresented. After adjusting for population size and prevalence of Internet access, the U.S. declined in influence (high absolute numbers but low relative numbers) while Europe, especially Northern Europe grow in importance. Other studies have confirmed the relatively high representation of European FLOSS participants, particularly from Germany and Finland [Dempsey et al. 2002; Tuomi 2004].

Motivations at the Individual Level. FLOSS teams are nothing without contributing participants and the question of what motivates participation has been a perennial 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; Lerner and Tirole 2005a] who argued that motivation was derived from indirect signaling about quality, with the payoff to come in higher career earnings. While it would be premature to argue that this work has been discredited, the empirical work on motivations has found little evidence for these expected motivations.

With regard to individual motivations, the empirical work is largely consistent. Studies have found that generally there are two types of motivations that propel individuals to contribute to FLOSS projects: extrinsic motivations and intrinsic motivations [Hars and Ou 2002]. Our analysis revealed that reputation [Hann et al. 2004] and reward motives [Hann et al. 2002] are the two most frequently mentioned extrinsic motivations. Enjoyment-based motivations such as fun [Ghosh 1998] and learning opportunity [Ye and Kishida 2003] are the two most commonly mentioned intrinsic ones. Many articles in this area employed a dependent variable of effort, or contribution, and are thus able to examine, through regression analysis, which forms of motivation drive the greatest effort. The use of a dependent variable allows studies to report motivations in two ways. The first is merely the presence, absence, or possibly ranking, of a factor in survey response [Hann et al. 2004]. The second is the correlation of a motivational factor with measures of effort [Hertel et al. 2003].

Lakhani and von Hippel [2003] argued that most FLOSS research on motivations has focused on the core tasks of developing, debugging and improving the software itself, little research on peripheral tasks they called "mundane but necessary task" (p.923). Using the Apache field support system as an example, the authors examined why some participants are motivated to contribute to this type of task. They found that 98% of the effort expended by information providers returns direct learning benefits to those providers.

Individual participation. A few studies have gone beyond reports of motives to examine 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] finds an average of 12.15 hours per week spent, with project leaders spending an average of 14.13 hours, and 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 spent 1.5 hours per week, dropping to just half an hour for frequent information seekers.

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. [2006], which studied 120 relatively successful SourceForge projects, found that the most common length of participation was no longer than a single month. The tenure of participants varies significantly according to their role. For example, Robles and Gonzalez-Barahona [2005] found quite long tenure amongst Debian package maintainers (more than half of the maintainers in 1998 continued to maintain their packages in 2005.)

Firm participation in FLOSS development. In addition to individuals, firms provide resources for open source software development. Companies are often involved in FLOSS development for such self-interested purposes as using open source software, building a service business around open source software, sponsoring open source software and having their engineers participate [Rossi and Bonaccorsi 2005; West and O'Mahony 2005]. Surveys have shown that as many as 45% of participants are paid by firms for their participation, either directly or indirectly2 [Hars and Ou 2002]. Research on this topic has also examined reasons for companies to invest internal resources in FLOSS development. For example, Bonaccorsi and Rossi [2005] found that firms are motivated to be involved with FLOSS because it allows smaller firms to innovate, because "many eyes" assist them in software development and because of the quality and reliability of FLOSS, with the ideological fight for free software coming at the bottom of the list. In comparison with individuals they find that firms focus less on economic motivations, though this is perhaps due to a self-report bias.

3.3.2 Project characteristics. Another input variable that has been considered within the FLOSS literature is project characteristics, that is, the distinguishing features of these projects. Software license types attract the most attention in this topic. Licenses are a particularly concrete differentiator of FLOSS projects. They play a crucial role with respect to all activities in FLOSS development, such as motivations, coordination, and relationships with commercial firms [Rossi 2004]. Licensing is also one of the most important tactics that used by a project to allow its intellectual property to be publicly and freely accessible and yet, governable [O'Mahony 2003].

One commonly discussed feature of a FLOSS license is its restrictiveness, which refers to two restrictions: copyleft provision (requires that modified versions of the software also be open) and viral provision (requires that the modified versions of the software be combined only with other programs distributed under licenses that share the first requirement) [Stewart and Gosain 2005]. Restrictive licenses refer to licenses that satisfy the above two provisions, while non-restrictive licenses do not. Lerner and Tirole

[2] Direct participation means that participating in an open source project is one of the participant's primary and explicitly stated job responsibilities, while indirect participation means that employees contribute to an open source project in fulfillment of other responsibilities to the firm.

[2005b] further divided FLOSS licenses into three classes according to its restrictiveness: unrestrictive (e.g., the Berkeley Software Definition (BSD) license), restrictive (i.e., licenses that satisfy copyleft provision, e.g., the FSF Lesser General Public License (LGPL)), and highly restrictive (i.e., licenses that satisfy both copyleft and viral provisions, e.g., the FSF General Public License (GPL)).

Though there is a clear focus of conceptual work on this topic, a few empirical studies have been conducted to examine the influence of license choices on various aspects of FLOSS development. For example, using data gathered from the Freshmeat website (www.freshmeat.net), Stewart and her colleagues investigate the impact of licensing restrictiveness on the success of FLOSS projects as indicated by popularity and vitality. The results partially support the finding that OSS projects that use a non-restrictive license became more popular over time than those that use a restrictive license. In another study of the impact of licensing choices on FLOSS success, using data from 62 projects in SourceForge (www.sourceforge.net), Colazo et al. [2005] found that copylefted projects were associated with more successful projects in terms of higher developer membership and productivity. Lerner and Tirole [2005b] investigated the relationship between project types and licensing choices. By examining the SourceForge database, the authors found that "projects geared toward end-users tend to have restrictive licenses, while those oriented toward developers are less likely to do so. Projects that are designed to run on commercial operating systems and whose primary language is English are less likely to have restrictive licenses. Projects that are likely to be attractive to consumers—such as games—and software developed in a corporate setting are more likely to have restrictive licenses. Projects with unrestricted licenses attract more contributors." (p.20).

3.3.3 Technology Use. The type of technology used by FLOSS teams is a very 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 discusses the importance of software version control systems such as CVS or Subversion, both for coordinating development and for mediating control over source code development when multiple developers may be working on any given portion of the code at once. This paper also discusses the interrelation of CVS use and email use (i.e. developers checking code into the CVS repository discuss the patches via email). Michlmayr [2004] illustrates the importance of bug trackers to coordinate among those working on questions and answers.

A small body of research studies the tools developers or users use to share and exchange knowledge. For example, Robbins [2002] discusses nine types of commonly used OSS engineering tools and illustrates their impact on the software development process. Using data from the developer mailing lists of two open-source software projects, Lanzara and Morner [2004] argue that technological artifacts and software-based artifacts are critical for knowledge sharing and creation in OSS development.

3.4 Processes

Processes represent dynamic interactions among FLOSS team members when they work on a project. Research on processes in FLOSS development has focused on software development practices as well as social processes within the projects.

3.4.1 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, it also mobilizes use of private resources [Scacchi 2004]. In the following sections, we consider in more detail articles reviewing specific practices, using the systems development lifecycle as an organizing structure.

Planning. It is commonly held that FLOSS projects do not engage in planning. For example, Glance [2004] examined the kernel change logs to determine the criteria applied for releasing the Linux kernel. He argues that a release contained whatever had been added, as opposed to there being 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].

Software Requirements Analysis. Similar to the planning stage, FLOSS projects do not conduct formal software requirements analyses. Scacchi [2004] states that FLOSS projects do not have conventional requirements documents. Instead, requirements are found in email messages, which emerge from discussions among users and developers about what the software should and should not do, and from after-the-fact assertions by developers about the need for a new piece of functionality. Similarly, Mockus et al. [2002] state that a user community can communicate its needs through bug reports or feature requests.

Coding. Much work in this section focuses on modularity and testing. Modularity has been seen as key to the feasibility of distributed development. Scacchi [2004] notes the importance of what he called "software extension mechanisms" that allow developers to easily add functionality to FLOSS projects via scripting or plug-ins.

Testing. There are mixed results for testing processes in FLOSS development, depending on different projects. Glance [2004] notes that the Linux kernel release process was not based on formal testing. Rather, it relies on individuals testing their own code before submission and subsequent testing by users of releases, also described as peer review [Glance 2004]. Stark [2002] notes that peer review has been used in conventional development as well and cites research showing that it works without meetings. Although this survey shows that only half of 23 FLOSS respondents said that their code was reviewed, one possible explanation is that it might be reviewed later, since it is open to inspection by any interested party. It also notes that any quality control approach relies on developer commitment to the process, which may come from compliance, identification or internalization, suggesting FLOSS relies on later mechanisms. Other studies have shown that some projects have more formal testing for releases. For example, Thomas [2005] describes several testing processes, such as the Linux Test Project, the Linux Stabilization Project. Dinh-Trong and Bieman [2005] notes that FreeBSD does have a more defined testing process.

Release. The nature of open code is that users can always access the latest version, which may or may not be stable, so it is difficult to speak of a single FLOSS approach to releases. Erenkrantz [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 might have quite informal release schedules. For example, Glance [2004] found that in Linux, releases came at an irregular rate. It is not clear what drove the schedule, but a release pattern was observed in terms of accepting patches for a while, then freezing acceptance of new code to allow the system to stabilize, though stability was assessed only by an absence of reported problems. In contrast, some projects have more organized ways of releasing. Dinh-Trong and Bieman [2005] report that FreeBSD releases a new version every four months. A "Release Engineering Team" coordinates the release, following a pattern similar to that of Linux. However, Raymond [1998] recommends "release early; release often". Projects tend to release incrementally, which suggests that every six months (to allow time to control bugs) is too infrequent.

Maintenance. Maintenance is a primary activity in FLOSS development, as it is in conventional development. In FLOSS development, however, the nature of maintenance is more like reinvention, which acts as "a continually emerging source of adaptation, learning, and improvement in FLOSS functionality and quality" [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 is often done commercially [Leibovitch 1999]. Other projects have commercial sponsors who sell support (MySQL, SugarCRM). Smaller projects rely on community support, e.g., via email or discussion boards. However, Singh et al. [2005] analyzed help interactions and found that this process was often inefficient because initial posts lacked the necessary information to answer questions, resulting in back-and-forth postings. The authors suggested that some details be captured automatically in initial reports, and also articulated the potential benefit of developing practices for more explicitly reusing information, e.g., marking particularly helpful answers to automatically populate a kind of FAQ.

3.4.2 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/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 participation in an open source project. The process of moving from a non-participant to a fully-fledged FLOSS developer has been addressed in a small volume of literature on socialization in FLOSS projects, that is, the strategies and processes of how 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 the social makeup of the project.

For example, in a study of socialization in the Freenet project, von Krogh et al [2003] propose that joining script (the level and type of activity a joiner goes through to become a member of the development process), specialization of new members, contribution barriers, and the feature gifts a new member can contribute are related to the process of being a new member. In concert with the findings of von Krogh et al (2003), Duchenaut [2003] studied socialization in the Python project from both learning and political perspectives, and found that participants who move to the center of a project, acting in a way that exposes more of the network to them, come to understand the relationships between people and code and, largely through action in the form of code, or detailed discussions of code, build legitimacy and "enroll allies" for their evolution towards the core of the project. He also highlights the manner in which the onus for socialization falls almost entirely on the developer, rather than the team. The process thus acts as a filter for participants that match the project, removing the need for recruitment effort typically required in the software industry.

Decision Making and Leadership. In conventional teams, decision-making effectiveness is very important to team effectiveness. A lack of transparency and consideration in the decision making process tends to alienate those who are not being consulted and erodes the sense of community [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 vs. 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] provides a decentralized decision making example by studying the GNOME project. Committees and task forces composed of volunteers are created to complete important tasks. Annual face-to-face meetings are also organized to make major decisions. By doing so, GNOME flattens the organizational structure of the project and allows broader participation in the decision-making process. In the Apache web server project, members adopt the voting mechanism to reach consensus [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 seen much discussion in the literature. Lerner and Triole [2002] identified the main duties of a leader in a FLOSS project as providing a vision; dividing the project into parts in which individuals can tackle independent tasks; attracting developers to the project; keeping the project together and preventing forking. 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. Individuals are perceived by others as leaders based on their sustained and strong technical contribution [Fleming and Waguespack 2005; Scozzi et al. 2008] and a structural position in their teams [Evans and Wolf 2005]. Leaders achieve this structural position by boundary spanning, which is conceptualized by Allen and Tushman as redirection of crucial information by well-respected guardians within and outside the team [Fleming and Waguespack 2005]. In their study of the Apache Lucene Java team, Scozzi et al. [2008] found that the founder of the projects also may hold a leadership position, since their opinions carry weight in the decision-making process.

Based on his participant observation in Apache FLOSS teams, Fielding [1999] describes how these teams exhibit shared leadership instead of having a single leader. According to Fielding, 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. Jensen and Scacchi [2005] suggest that the volunteer nature of these teams affect responsibility and accountability within the teams. Thus, leadership is not asserted until a community member volunteers to champion a cause [Jensen and Scacchi 2005].

Crowston and colleagues [2007], based on their earlier theoretical work [Crowston et al. 2005a], conducted a further study to explain how leadership is manifested in FLOSS teams. Their preliminary findings, based on two FLOSS teams, suggest that the more successful team showed decentralized leadership in the areas of task coordination, substantive task contribution, boundary spanning and group maintenance. Supporting their earlier proposition, the researchers also showed that the task of developing team structures was more centralized: The more successful team was centralized in this area, and the less successful team was centralized initially, yet as the team became less successful, developing team structures was more decentralized.

Howison et al. [2006] studied how the social structure of FLOSS teams change over time by analyzing communications related to enhancements and bug-fixing activity using social network analysis. The researchers identified the individuals who took a central position in this type of communication as leaders. This study explores how leaders may change over time and new leaders emerge, suggesting that leadership is not only shared but also emergent in these teams.

Coordination/collaboration. Coordination manages dependencies between activities [Malone et al. 1999]. The FLOSS environment makes coordination more difficult for several reasons. Volunteers without formal contracts, dispersed contributors, the virtual environment, and different types of actors (firm-sponsored vs. volunteers) all complicate coordination efforts [van Wendel de Joode and Egyedi 2005]. Coordination activities play an important role in FLOSS development, and is critical to project success [Sagers 2004] and to sustaining collective efforts in FLOSS teams, especially for large project such as Lunix [Kuwabara 2000].

The backgrounds and characteristics of the different projects may influence the use of coordination mechanisms in general. For example, Java uses ex ante mechanisms while Linux uses ex post mechanisms, because Java is a company sponsored project while Linux is basically a community project [van Wendel de Joode and Egyedi 2005]. Yamauchi et al. [2000] makes a similar observation about the ex post mechanism: Coordination follows action. Developers choose to act without first declare commitment.

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

Mechanisms to control the number of developers. The general collaborative mode of FLOSS development is that a small portion of developers are responsible for most of the output [Koch and Schneider 2002; Mockus et al. 2002; Crowston and Scozzi 2004; Dinh-Trong and Bieman 2005]. Based on a theoretical framework of network governance, Sagers [2004] demonstrates that restricted access to the development team improves coordination within the project.

Modularity and division of labor. Modularity is the most explicit mechanism used in FLOSS development. It keeps the core software product small enough to be handled by a small core group, and makes communication smooth and effective [Mockus et al. 2002; Jensen and Scacchi 2005]. 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] notes that while developers tend to work on the same modules repeatedly, most modules were worked on by several people, which does not support the notion of individual code ownership and requires other ways of coordinating. One possible way is to introduce coordinators to coordinate development between modules, as suggested by Asklund and Bendix [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 FLOSS development [Mockus et al. 2000; Mockus et al. 2002; Crowston and Scozzi 2004; Crowston et al. 2005c].

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 Jooode [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 growing body of research recognizing that OSS development faces knowledge management (KM) challenges because of its highly distributed, knowledge intensive characteristics [Ciborra and Andreu 2001; Edwards 2001; Becking et al. 2005]. This body of research focuses on knowledge creation, knowledge/information sharing and knowledge reuse [Dafermos 2001; Lakhani and von Hippel 2003; Lee and Cole 2003; Hemetsberger and Reinhardt 2004; Lanzara and Morner 2004; Huysman and Lin 2005; Raisinghani 2005; von Krogh et al. 2005]. For example, Huysman and Lin [2005] uses social worlds theory to study how Linux users communicate with each other virtually to solve their problems. The authors find that problem solving involves continuous negotiations and redefinition on a locally-found problem. They also find that online communities without strict membership requirements activate cross-boundary learning and knowledge sharing, such as between interested Linux users coming from different social worlds. Based on the analysis of developer mailing lists of two large-scale open source projects, Lanzara and Morner [2004] illustrate how the processes of knowledge making and sharing are supported by dense social interaction and by the peculiar organizing features inscribed in technological artifacts. Also shown is that the evolutionary features of knowledge making and sharing in virtual environments challenge current ways of conceptualizing knowledge processes within and across organizations. Von Krogh et al. [2005] report on the reuse of knowledge in software development based on 15 open source projects. The authors find that the effort to search, integrate and maintain external knowledge influences the form of knowledge to be reused.

Much research uses learning theory as its theoretical perspective and draws on communities of practice literature to conceptualize how to share and create knowledge online. Using the case of the Linux kernel development project, Lee and Cole [2003] build a model of community-based, evolutionary knowledge creation to study how thousands of talented volunteers, dispersed across organizational and geographical boundaries, collaborate via the Internet to produce a knowledge-intensive, innovative product of high quality. The model suggests that the product development process can be effectively organized as an evolutionary process of learning driven by criticism and error correction. Hemetsberger and Reinhardt [2004] take a social view of learning and knowledge creation to demonstrate how online communities of practice successfully overcome the problem of tacit knowledge transformation through technological tools, task-related features, collective reflection, stories and usage scenarios.

3.5 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.

3.5.1 Trust. We first examine research that considers project team trust. Trust has been studied extensively in group research, and has been noted as a determining factor to achieve the effectiveness of team collaboration. Researchers have also suggested that trust is important in FLOSS team development [Stewart and Gosain 2001; Evans and Wolf 2005]. Trust is often related to team effectiveness. For example, Stewart and Gosain [2001] propose that shared ideology enables the development of affective and cognitive trust, which in turn lead to group efficacy and effectiveness. In a study of leadership in FLOSS teams, Fleming and Waguespack [2005] argue that successful leaders are the product of strong technical contribution and a structural position (brokerage and boundary spanning) that can bind the community together. But the effectiveness of brokerage is contingent on trust. An inherent lack of trust associated with brokerage positions can be overcome through physical interaction or contributions within technological boundaries. But not all researchers share the same belief. In a study of published case studies of

FLOSS projects, Gallivan [2001] found that group effectiveness can be achieved in the absence of trust if a set of control and self control mechanisms is presented.

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

Roles. In an emergent context, something as seemingly simple as role becomes more complex. While in one context and one time, a participant may be a 'core developer', in another context they may be a support question asker [Ye and Kishida 2003]. Most research on roles focuses on the differences between distinct roles and how to define roles. Gacek and Arief [2004] suggest distinguishing between passive and active users, and among active users, between non-developers and developers, and among developers between co-developers and core developers, with corresponding increases in responsibility and contribution.

Alternative methods have been used to examine the sizes of the core and periphery groups. Crowston et al. [2006] studied the distribution of contributions to the bug-tracking system, for 120 Sourceforge teams using three different methods: first, the self-reporting of teams based on the list of developers on the Sourceforge site; the second, core/periphery measures using the social network structure; and third, an analogy to Bradford's law about the distribution of academic publications. The measures indicated that the developer's list was not a good indicator of core membership (at least in bug-fixing) and that the skew of contribution in the communications domain was substantially higher than in the code domain. The more reliable measures pegged the core group size at a median of 3 (about 5% of participants in the project). The results are basically in line with early results in sociology, such as James [1951].

Lin [2006] describes interviews she conducted with firms involved in open source and with developers that had moved from community involvement to working for a company, but still doing open source. She found that developers working inside companies have hybrid roles, such that they can draw on resources from the firm and the community, but have to balance their loyalties and act as translators in situations of different aims.

Level of Commitment. Researchers have also been interested in the distribution of different types of effort, such as code contribution [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) had contributed over 83% of modification requests and 66% of problem reports, which is lower but still quite high. They compared these contribution distributions to commercial projects and found them to be higher, sometimes substantially so, suggesting that while FLOSS projects have larger total numbers of contributors, the bulk of activity, especially for new features, is quite highly centralized. Efforts to replicate these findings have tended to show a smaller differential in the distributions. Dinh-Trong [2005], in a study of the FreeBSD project, found that the top 15 contributors (of 354) contributed only 57% of new source code and one needed the top 47 to reach the 80% figure. Bug fixing was again found to be more widely distributed, with the top 15 developers checking in only 40% of the changes. Dinh-Trong did observe that these statistics conflate the effort over the entire lifetime of the project and recalculate the measures with three year windows, but still find 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, it is believed 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] relates that all the core participants in Apache had other “real jobs”, which usually involved using the software for which they were a contributor. Lakhani and von Hippel [2003] finds 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 was considered spare time by participants and those who consider they have more spare time are likely to spend more of it developing FLOSS, although the strength of the relationship fell as spare time continued to rise (decreasing returns). This finding is supported by Michlmayr [2004], who reported that participants understand others to have “real” jobs that justifiably interfere with their participation and by Crowston et al. [2005b] who report that participants at face-to-face conferences cite the ability to spend long blocks of relatively uninterrupted time focusing on a FLOSS project as a major benefit of attendance.

Shared Mental Models. Prior research suggests that the existence of accurate shared mental models that guide member actions are important for team effectiveness [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. [1990-1991], note that, "a fundamental problem in building large systems is the development of a common understanding of the requirements and design across the project team." They go on to say that, "the transcripts of team meetings reveal the large amounts of time designers spend trying to develop a shared model of the design". Scozzi et al. [2008] analyzed mental models in a FLOSS project using cognitive mapping and process analysis. Specifically, they compared the models of four developers from the Apache Lucene Java project. Their analysis suggests that there is a high level of sharing among core developers on aspects such as key definitions (e.g., project goals, users and challenges) and some aspects of the cause maps, but the sharing was not complete, with some differences related to tenure and role in the project.

3.6 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.

3.6.1 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. Crowston and his colleagues [2003] propose a range of measures to evaluate FLOSS team/project success. Based on a combination of literature review, a consideration of the OSS development process, and an analysis of the OSS developers' opinions, they identified 7 measures of FLOSS project success: four from the IS literature review (system and information quality, user satisfaction, use, individual and organizational impacts) and three from examination of OSS development (project output, process and outcomes for project members). In another paper, collecting data from 122 projects hosted on SourceForge, Crowston et al. [2004] test validity and utility of four indicators from these 7 measures (number of members of the extended development community, project activity, bug fixing time and number of downloads). In this paper, these 7 measures are used to reorganize the papers we collected. Although FLOSS success is a multidimensional construct, most research only used one dimension to assess success.

The most frequently used measure of success emerging from these studies is system and information quality (26 out of 37). Although theory has described indicators such as code quality and documentation quality to measure FLOSS system and information quality [Crowston et al. 2003], the majority of the empirical work has focused on using code quality measures. This body of literature provides a variety of indicators to measure code quality such as maintainability [Bezroukov 1999; Hecker 1999; Schach et al. 2003a; Schach et al. 2003b; Samoladas et al. 2004], product/software quality [Stamelos et al. 2002; Glance 2004; Michlmayr 2004; Schmidt 2004; Gyimothy et al. 2005], defect density [Mockus et al. 2000; Paulson et al. 2004], usability [Nichols et al. 2001; Schmidt 2004; Hanson et al. 2005], reliability [Leibovitch 1999; Harris 2004; Gyimothy et al. 2005; van Wendel de Joode and de Bruijne 2006], and value of software output [Harris 2004]. Further, some studies compare the quality of open source software with propriety software and the results are mixed. For example, by comparing three closed source software projects and three open source software projects, Paulson et al. [2004] found that generally OSS has fewer defects than closed source software. But Stamelos et al. [2002] offered a structural code analysis and suggest that the code quality of an OSS may be higher than which someone against OSS might expect, but lower than the quality implied by the industrial standard.

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, most of which focus on specific project characteristics such as software component, project types, project life cycles, sponsorships and license types. For example, Stamelos et al. [2002] found a partial negative relationship between component size and user satisfaction by examining a sample of 100 C programs found in the SUSE Linux 6.0 release. 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). They also find that increases in vitality (which refers to how much developer effort and attention is expended on the project) had a significant positive effect on popularity. Based on social movement theory, Colazo et al. [2005] performed an exploratory study using data from 62 relevant OSS projects spanning an average of three years of development time. The results indicate that copyleft projects were associated with higher developer membership and productivity. Chawla et al. [2003] found that level of support activity and communications in SourceForge were predictors of page views and downloads.

Some studies reported relationship between processes and team effectiveness. Long and Yuan [2005] report the important role of core developers in projects' success. By statistically examining 300 OSS projects, they demonstrate that core developers' leadership and project advocacy are crucial in determining the fate of the OSS projects. Franke and Hippel [2003] found that users are more satisfied when they can modify their own software.

A few studies investigate the impact of emergent state factors on project performance. For example, using content analysis to examine a set of published case studies of OSS projects, Gallivan [2001] noted that although trust is rarely mentioned, ensuring control is an important criterion for effective performance within OSS projects. Wynn [2003] found the fit between the life cycle stage and the specific organizational characteristics of these projects (focus, division of labor, role of the leader, level of commitment, and coordination/control) is an indicator of the success of a project as measured by the satisfaction and involvement of both developers and users.

3.6.2 Open source software implementation. Outside of the main stream of most FLOSS research, some studies also paid attention to how OSS is being adopted and used in different contexts [Fitzgerald and Kenny 2003; Bleek and Finck 2004; Fitzgerald and Kenny 2004; Vemuri and Bertone 2004; Goode 2005; Holck et al. 2005; Miralles et al. 2005; Waring and Maddocks 2005; Yan et al. 2005].

Dinkelacker et al. [2002] describe activities at Hewlett Packard that aim to adapt the benefits of open source software for internal use, through the progressive introduction of open source practices. They began with "inner source," the opening of all code behind the corporate firewall, then "controlled source" which restricts access to contracted partners and finally "open source," where 'the community' in general is invited to participate. Chan [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.

3.6.3 Evolution. This section discusses FLOSS evolution over time. The literature has focused on two aspects: the evolution of the product, which is the software developed in this context; and the evolution of the community to that develops and maintains the software. Different types of FLOSS projects have different patterns of system evolution and community evolution [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 did not obey Lehman's laws which say that "as the system grew, the rate of growth would slow, with the system growth approximating an inverse square curve".

Some literature looks in detail at code evolution patterns. Scacchi [2004] notes that code tends to evolve incrementally rather than change radically. Capiluppi [2004] found an unbalanced evolution patterns for some codes in an OSS project called ARLA – "some [code] branches may appear more appealing than others, and are extensively evolved, while other[s] remain in the same status for all the life cycle". Antoniol et al. [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] discuss the impact of member retirement on community structure. They argued that a snowball effect might lead more members to leave when one member drops out, which might result in network separation and disintegration. It is important to maintain a balanced composition of all the different roles in a community [Nakakoji et al. 2002], but it does not mean that the roles are fixed. Each member can play a larger role if he or she contributes more and is acknowledged by others [Ye and Kishida 2003].

Of course, code and community do not exist separately. They co-evolve and have an impact on each other. Nakakoji et al. [2002] argues that the contribution made by members is the source of system evolution, and the system evolution in turn affects the contribution distribution among the developers, and thus redefines the roles of the contributors. Similarly, Capiluppi [2004] argues that “when the tree structure reaches some status, the process of joining as a core developer seems to forestall” (p.23).

4. CONCLUSIONS

In the previous section we reviewed the empirical research on FLOSS development and use in an effort to assess the state of the literature. From the analysis we can see that we are still in the early stages of investigation of FLOSS and significant empirical work remains to understand this phenomenon. For the period 1998 to early 2006 a slightly larger number of empirical papers were published in conferences than in journals. There are also relatively equal numbers of empirical papers and non-empirical papers, which might indicate the relative newness of this area.

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-moderator-output-input (IMOI) model that was used in section 3. We also address a number of methodological and theoretical issues related to the study of FLOSS.

4.1 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] reports that future career prospects were significantly related to planned future activities, but not significantly related to actual contribution, suggesting that this motive might provide initial drive but go unrealized. It would be an interesting finding to discover whether participants with particular types of motivation are more likely to continue to contribute or to achieve leadership roles. Further, few studies have examined changes in motivation over time. Previous research has indicated that motivations may alter over time. For example, Ghosh [2002] mentions that reputation was a stronger motivation amongst the longer term participants (who might be expected to actually have garnered reputation). But these analyses are preliminary and longitudinal analyses are needed to examine the phenomenon in detail. This is particularly important for insight on the rise of bounties (e.g., Gnome) and temporary corporate support of participants (e.g., Google's Summer of Code).

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

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

4.2 Processes

Previous research in this area has focused on examining mechanisms used in different processes as described in Figure 5. More research is needed on factors that affect processes and how the characteristics of FLOSS development influence processes. For example, few studies have touched on the impact of team diversity (e.g., team members' motivations, values or skills) on their collaboration. Further, how do external environmental factors, such as project type, company sponsored versus non-sponsored, interact with team and project development processes?

Social processes represent an area in which major gaps exist in the FLOSS research literature. In particular, little research has been conducted on social processes related to conflict management and team maintenance. Conflicts sometimes can have significant negative effects on FLOSS development, given its virtual and self-organizing nature.

Team maintenance encompasses the pro-social, discretionary, and relation-building behaviors that create and maintain reciprocal trust and cooperation between members [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 [Morgan 2000], but little research has been done on this topic in the FLOSS research literature. Several questions remain unanswered. What kind of factors likely trigger conflict? What's the role of leaders in conflict management? How team maintenance is created and sustained over time? Is there any relationship between project types and team maintenance behaviors?

Another possible important factor is how projects manage the knowledge necessary for a successful development effort. Researchers have recently begun to examine knowledge management in FLOSS. 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 this area on this topic in order to understand how members integrate knowledge from different sources. In particular, what mechanisms and team norms are used to store knowledge contributed by team members? What techniques are used to identify useful knowledge given the huge information flow?

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

4.3 Emergent States

As stated, there is little discussion of team members' interaction patterns over time. For example, roles are probably best studied as structurationally emergent, 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? What kinds of factors trigger these changes? Are there any patterns associated with different projects, and different project development phases? What is the relationship between processes and emergent states development?

4.4 Outputs

Most current research on measuring 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 missing. Since FLOSS development is usually a long-term project, it is important to include this measure in evaluating FLOSS effectiveness.

Another issue is that the link from output to input has not been addressed in previous literature. In a FLOSS developmental sequence, outputs at a time are 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], little empirical studies have been done in FLOSS research. More research is needed on how outputs contribute/change inputs. For example, how do outputs such as user satisfaction impact team structure in the future?

4.5 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 (e.g., drawn from SourceForge). The use of this type of data is not without problems. SourceForge provides 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]. Archive data may also have a high omission rate [Chen et al. 2004], so more data sources are needed.

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

A third methodological concern with current FLOSS research is the sampling strategies used. For example, most research has studied well-established projects, not projects in the initial or transition phases. Most research has studied successful projects, not unsuccessful projects. Most research has studied only a few projects (usually less than 10 and often only one). Also, there has been insufficient attention to segmenting research by types of projects, e.g., based on the complexity of the project or the intended audience. Future research needs to include projects in different phases and types in order to advance our understanding of FLOSS development. Studies should also attempt to advance the frontier of research designs shown in Figure 2 by simultaneously studying larger samples of projects in order to generalize the findings and studying projects in more depth, with richer data.

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

A final concern is with the paucity of longitudinal studies in FLOSS research. As the IMOI model suggests, team interactions are probably best studied as a time series, since their effects (such as coordination costs) should be felt only if the interactions actually occur in the same timeframe, which cross-sectional measures may not show.

In conclusion, our aim in this article is to synthesize the empirical work done to date in order to clarify what we know and do not know about FLOSS development and to suggest promising directions for further work. Empirical research on FLOSS is still in its early stage and shows tremendous promise for future research. To stimulate such work, we have presented a set of research questions around IMOI model, which we believe provide a further step to understand the FLOSS phenomenon.

REFERENCES

ALHO, K., SULONEN, R. 1998. Supporting virtual software projects on the Web. *The 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Palo Alto, CA, USA. June 17-19, 1998.

ANTONIOL, G., VILLANO, U., MERLO, E., PENTA, M. D. 2002. Analyzing cloning evolution in the Linux kernel. *Information and Software Technology* **44**(13), 755.

ASKLUND, U., BENDIX, L. 2001. Configuration Management for Open Source Software. *In Proceedings of 1st Workshop on Open Source Software Engineering*. Toronto, Ontario, Canada. May 15, 2001.

BECKING, J., COURSE, S., ENK, G. V., HANGYI, H. T., ET AL. 2005. MMBase: An open-source content management system. *IBM Systems Journal* **44**(2), 381.

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

BLEEK, W.-G., FINCK, M. 2004. Migrating a Development Project to Open Source Software Development. *Proceedings of the ICSE 4th Workshop on Open Source Software Engineering*. Edinburgh, Scotland, United Kingdom. May 25, 2004.

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

CAPILUPPI, A. 2004. Improving comprehension and cooperation through code structure. *Proceedings of the ICSE 4th Workshop on Open Source Software Engineering*. Edinburgh, Scotland, United Kingdom. May 25, 2004.

CHAN, A. 2004. Coding Free Software, Coding Free States: Free Software Legislation and the Politics of Code in Peru. *Anthropological Quarterly* **77**(3), 531-545.

CHAWLA, S., ARUNASALAM, B., DAVIS, J. 2003. Mining Open Source Software (OSS) data using Association Rules Network. *Advances in Knowledge Discovery and Data Mining.* **2637,** 461-466.

CHEN, K., SCHACH, S. R., YU, L. G., OFFUTT, J., HELLER, G. Z. 2004. Open-source change logs. *Empirical Software Engineering* **9**(3), 197-210.

CHIDAMBARAM, L., BOSTROM, R., WYNNE, B. 1990-1991. A Longitudinal Study of the Impact of Group Decision Support Systems on Group Development. *Journal of Management Information Systems* **7**(3), 7-25.

CIBORRA, C. U., ANDREU, R. 2001. Sharing knowledge across boundaries. *Journal of Information Technology* **16**(2), 73-81.

COLAZO, J. A., FANG, Y., NEUFELD, D. J. 2005. Development Success in Open Source Software Projects: Exploring the Impact of Copylefted Licenses. *Americas Conference on Information Systems (AMCIS 2005)*. Omaha, Nebraska, USA. August 11-14, 2005.

COLLLINS, J., DRUCKER, P. 1999. A Conversation between Jim Collins and Peter Drucker. Drucker Foundation News. **7:** 4–5.

CROWSTON, K., ANNABI, H., HOWISON, J. 2003. Defining Open Source Software project success. *Proceedings of the 24th International Conference on Information Systems (ICIS 2003)*.

CROWSTON, K., ANNABI, H., HOWISON, J., MASANO, C. 2004. Towards a Portfolio of FLOSS Project Success Measures. *Proceedings of the ICSE 4th Workshop on Open Source Software Engineering*. Edinburgh, Scotland, United Kingdom. May 25, 2004.

CROWSTON, K., HECKMAN, R., ANNABI, H., MASANGO, C. 2005a. A structurational perspective on leadership in Free/Libre Open Source Software teams. *Presentation at the First International Conference on Open Source Systems*. Genova, Italy. July 11-15, 2005.

CROWSTON, K., HECKMAN, R., MISIOLEK, N., ESERYEL, U. Y. 2007. Emergent leadership in self-organizing virtual teams. . *Twenty Eighth International Conference on Information Systems*. Montreal, Canada. December 9-12, 2007.

CROWSTON, K., HOWISON, J. 2005. Hierarchy and centralization in Free and Open Source Software team communications. *Knowledge, Technology and Policy* **18**(4), 65-85.

CROWSTON, K., HOWISON, J., MASANGO, C., ESERYEL, U. Y. 2005b. Face-to-face interactions in self-organizing distributed teams. *Academy of Management Conference*. Honolulu, Hawaii, USA. August 5-10, 2005.

CROWSTON, K., SCOZZI, B. 2004. Coordination practices for bug fixing within FLOSS development teams. *Presentation at 1st International Workshop on Computer Supported Activity Coordination*. Porto, Portugal. April 13-14, 2004.

CROWSTON, K., WEI, K., LI, Q., ESERYEL, U. Y., HOWISON, J. 2005c. Coordination of free/libre open source software development. *International Conference on Information Systems*. Las Vegas, NV, USA. December 11-14, 2005.

CROWSTON, K., WEI, K., LI, Q., HOWISON, J. 2006. Core and periphery in Free/Libre and Open Source software team communications. *Hawai'i International Conference on System System (HICSS-39)*.

DAFERMOS, G. 2001. Management and Virtual Decentralized Networks: The Linux Project, opensource.mit.edu.

DEMPSEY, B. J., WEISS, D., JONES, P., GREENBERG, J. 2002. Who is an open source software developer? *Communications of the Acm* **45**(2), 67-72.

DINH-TRONG, T. T., BIEMAN, J. M. 2005. The FreeBSD project: A replication case study of open source development. *Ieee Transactions on Software Engineering* **31**(6), 481-494.

DINKELACKER, J., GARG, P. K., MILLER, R., NELSON, D. 2002. Progressive Open Source. *Proceedings of ICSE '02*. Orlando, FL, ACM.

DUCHENEAUT, N. 2003. The reproduction of Open Source Software programming communities.

EDWARDS, K. 2001. Epistemic communities, situated learning and Open Source Software development. *Epistemic Cultures and the Practice of Interdisciplinarity Workshop.*

ERENKRANTZ, J. R. 2003. Release Management Within Open Source Projects. *Proceedings of the ICSE 3rd Workshop on Open Source Software Engineering*. Portland, Oregon, USA. May 3, 2003.

EVANS, P., WOLF, B. 2005. Collaboration rules. *Harvard Business Review* **83**(7), 96-103.

FIELDING, R. T. 1999. Shared leadership in the Apache Project. *Association for Computing Machinery. Communications of the ACM* **42**(4), 42.

FITZGERALD, B. 2006. The Transformation of Open Source Software. *MIS Quarterly* **30**(3), 587-598.

FITZGERALD, B., KENNY, T. 2003. Open Source Software can Improve the Health of the Bank Balance - The Beaumont Hospital Experience.

FITZGERALD, B., KENNY, T. 2004. Developing an information systems infrastructure with open source software. *Ieee Software* **21**(1), 50-+.

FLEMING, L., WAGUESPACK, D. 2005. Penguins, Camels, and Other Birds of a Feather: Brokerage, Boundary Spanning, and Leadership in Open Innovation Communities, opensource.mit.edu.

FRANKE, N., VON HIPPEL, E. 2003. Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software. *Research Policy* **32**(7), 1199-1215.

GACEK, C., ARIEF, B. 2004. The many meanings of Open Source. *IEEE Software* **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. *Information Systems Journal* **11**(4), 277-304.

GARRISON, R., ANDERSON, T., ARCHER, W. 2000. Critical thinking in a text-based environment: Computer conferencing in higher education. *The Internet and Higher Education* **2**, 87-105.

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

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. 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 Information and Communication Technologies (ICT) sector in the EU, UNU-Merit.

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

GODFREY, M. W., TU, Q. 2000. Evolution in open source software: A case study. *2000 International Conference on Software Maintenance*. San Jose, CA, USA. October 11-14, 2000.

GOODE, S. 2005. Something for nothing: management rejection of open source software in Australia's top firms. *Information & Management* **42**(5), 669-681.

GYIMOTHY, T., FERENC, R., SIKET, I. 2005. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Transactions on Software Engineering* **31**(10), 897+.

HACKMAN, J. R., 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. New York, Academic Press**,** 45-99.

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

HANN, I.-H., ROBERTS, J., SLAUGHTER, S. A. 2004. Why developers participate in open source software projects: An empirical investigation. *Twenty-Fifth International Conference on Information Systems*.

HANSON, V. L., BREZIN, J. P., CRAYNE, S., KEATES, S., ET AL. 2005. Improving Web accessibility through an enhanced open-source browser. *IBM Systems Journal* **44**(3), 573.

HARRIS, J. S. 2004. Mission-critical development with open source software: Lessons learned. *Ieee Software* **21**(1), 42.

HARS, A., OU, S. S. 2002. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce* **6**(3), 25-39.

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

HEMETSBERGER, A., REINHARDT, C. 2004. Sharing and Creating Knowledge in Open-Source Communities: The case of KDE. *The Fifth European Conference on Organizational Knowledge, Learning, and Capabilities*. Innsbruck, Austria. April 2-3, 2004.

HERTEL, G., NIEDNER, S., HERRMANN, S. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy* **32**(7), 1159-1177.

HOLCK, J., LARSEN, M. H., PEDERSEN, M. K. 2005. Managerial and technical barriers to the adoption of open source software. *Cots-Based Software Systems, Proceedings*. **3412,** 289-300.

HOWISON, J., CROWSTON, K. 2004. The perils and pitfalls of mining SourceForge. *Presentation at the Workshop on Mining Software Repositories, 26th International Conference on Software Engineering*. Edinburgh, Scotland, United Kingdom. May 25, 2004.

HUYSMAN, M., LIN, Y. 2005. Learn to solve problems: a virtual ethnographic case study of learning in a GNU/Linux Users Group. *eJOV - The Electronic Journal for Virtual Organizations and Networks* **7**, 56-69.

ILGEN, D. R., HOLLENBECK, J. R., JOHNSON, M. 2005. Team in Organizations: From Input-Process-Output Models to IMOI models. *Annual Review of Psychology* **56**, 517-543.

JAMES, J. 1951. A preliminary study of the size determinant in small group interaction. *American Sociological Review* **16**(4), 474-477.

JENSEN, C., SCACCHI, W. 2005. Collaboration, Leadership, Control, and Conflict Negotiation in the Netbeans.org Open Source Software Development Community. *Proceedings of the Hawai'i International Conference on System Science (HICSS)*. Big Island, Hawai'i.

KOCH, S. 2004. Profiling an Open Source Project Ecology and Its Programmers. *Electronic Markets* **14**(2), 77-88.

KOCH, S., SCHNEIDER, G. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal* **12**(1), 27-42.

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

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

LANCASHIRE, D. 2001. The fading altruism of Open Source development. *First Monday* **6**(12).

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

LEE, G. K., COLE, R. E. 2003. From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science* **14**(6), 633-649.

LEIBOVITCH, E. 1999. The business case for Linux. *IEEE Software* **16**(1), 40-44.

LERNER, J., TIROLE, J. 2002. Some simple economics of open source. *Journal of Industrial Economics* **50**(2), 197-234.

LERNER, J., TIROLE, J. 2005a. The economics of technology sharing: open source and beyond. *Journal of Economic Perspectives* **19**(2), 99-120.

LERNER, J., TIROLE, J. 2005b. The Scope of Open Source Licensing. *Journal of Law Economics & Organization* **21**(1), 20-56.

LIN, Y. 2006. Hybrid Innovation: How Does the Collaboration Between the FLOSS Community and Corporations Happen? *Knowledge, Technology and Policy* **18**(4), 86-100.

LONG, J., YUAN, M. J. 2005. Are all Open Source Projects Created Equal? Understanding the Sustainability of Open Source Software Development Model. *Americas Conference on Information Systems*. Omaha, Nebraska, USA. August 11-15, 2005.

LUTHIGER STOLL, B. 2005. Fun and Software Development, opensource.mit.edu.

MALONE, T. W., CROWSTON, K., LEE, J., PENTLAND, B., DELLAROCAS, C., WYNER, G., QUIMBY, J., OSBORN, C. S., BERNSTEIN, A., HERMAN, G., KLEIN, M., O'DONNELL, E. 1999. Tools for inventing organizations: Toward a handbook or organizational processe. *Management Science* **45**(3), 425-443.

MARKS, M. A., MATHIEU, J. E., ZACCARO, S. J. 2001. A Temporally Based Framework and Taxonomy of Team Processes. *Academy of Management Review* **26**(3), 356-376.

MARTINS, L. L., GILSON, L. L., MAYNARD, M. T. 2004. Virtual Teams: What do We Know and Where do We Go from Here? *Journal of Management* **30**(6), 805-835.

MCGRATH, N. 2004. What exactly are the merits of open-source software? Microsoft and Novell go head to head to talk out the issues - View#1. *Iee Review* **50**(10), 46-48.

MICHLMAYR, M. 2004. Managing Volunteer Activity in Free Software Projects, opensource.mit.edu.

MIRALLES, F., SIEBER, S., VALOR, J. 2005. CIO Herds and User Gangs in the Adoption of Open Source Software. *European Conference on Information Systems (ECIS 2005)*. Regensburg, Germany. May 26-28, 2005.

MOCKUS, A., FIELDING, R. T., HERBSLEB, J. D. 2000. A case study of Open Source Software development: The Apache server. *Proceedings of the International Conference on Software Engineering (ICSE'2000).*

MOCKUS, A., FIELDING, R. T., HERBSLEB, J. D. 2002. Two case studies of open source software development: Apache and Mozilla. *Acm Transactions on Software Engineering and Methodology* **11**(3), 309-346.

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

MORAND, D. A., 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 (HICSS' 03)*. Hilton Waikoloa Village, Hawaii. January 6-9, 2003.

MORGAN, E. L. 2000. The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. *Information Technology and Libraries* **19**(2), 105.

NAKAKOJI, K., YAMAMOTO, Y., NISHINAKA, Y., KISHIDA, K., YE, Y. 2002. Evolution patterns of open-source software systems and communities. 85.

NICHOLS, D. M., THOMSON, K., YEATES, S. A. 2001. Usability and open-source software development. *Proceedings of the Symposium on Computer Human Interaction*. Palmerston North, New Zealand. July 6, 2001.

O'LEARY, M. B., CUMMINGS, J. N. 2007. The Spatial, Temporal, and Configurational Characteritics of Geographic Dispersion in Teams. *MIS Quarterly* **31**(3), 433-452.

O'MAHONY, S. 2003. Guarding the commons: how community managed software projects protect their work. *Research Policy* **32**(7), 1179-1198.

OH, W., JEON, S. 2004. Membership Dynamics and Network Stability in the Open-Source Community: The Ising Perspective. *Proceedings of International Conference on Information Systems 2004*. Washington DC, USA. December 12-15, 2004.

PAULSON, J. W., SUCCI, G., EBERLEIN, A. 2004. An empirical study of open-source and closed-source software products. *Ieee Transactions on Software Engineering* **30**(4), 246-256.

POWELL, A., PICCOLI, G., IVES, B. 2004. Virtual Teams: a review of current literature and directions for future research. *The DATA BASE for Advances in Information Systems* **35**(1), 6-36.

RAISINGHANI, M. S. 2005. Search Engine Technology: A Closer Look at Its Future. *Information Resources Management Journal* **18**(2), I.

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*. New York, Viking.

ROBBINS, J. E. 2002. Adopting OSS Methods by Adopting OSS Tools. *Proceedings of the ICSE 2nd Workshop on Open Source Software Engineering*. Orlando, Florida, USA. May 25, 2002.

ROBLES, G., GONZALEZ-BARAHONA, J. M. A. M. M. 2005. Evolution of volunteer participation in libre software projects: Evidence from Debian. *Proceedings of the First International Conference on Open Source Systems*. Genova, Italy. July 11 - 15, 2005.

ROSSI, C., BONACCORSI, A. 2005. Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach?, opensource.mit.edu.

ROSSI, M. A. 2004. Decoding the "Free/Open Source (F/OSS) Software Puzzle": A survey of theoretical and empirical contributions, Universiti degli Studi di Siena, Dipartimento Di Economia Politica.

SAGERS, G. W. 2004. The influence of network governance factors on success in open source software development projects. *Proceedings of International Conference on Information Systems 2004*. Washington, DC, USA. December 12-15, 2004.

SAMOLADAS, I., STAMELOS, I., ANGELIS, L., OIKONOMOU, A. 2004. Open source software development should strive for even greater code maintainability. *Communications of the Acm* **47**(10), 83-87.

SCACCHI, W. 2002. Understanding the requirements for developing Open Source Software systems. *IEE Proceedings Software* **149**(1), 24–39.

SCACCHI, W. 2004. Free/Open Source Software Development Practices in the Computer Game Community. *IEEE Software* **21**(1), 56-66.

SCACCHI, W. 2007. Free and open source software development: Recent research results and methods. *Advances in Computers*. M. Zelkowitz, Elsevier Press. **69,** 243-295.

SCHACH, S. R., JIN, B., WRIGHT, D. R., HELLER, G. Z., OFFUTT, A. J. 2003a. Determining the Distribution of Maintenance Categories: Survey versus Measurement. *Empirical Software Engineering* **8**(4), 351-365.

SCHACH, S. R., JIN, B., WRIGHT, D. R., HELLER, G. Z., OFFUTT, A. J. 2003b. Maintainability of the Linux Kernel. **2003**.

SCHMIDT, D. P. 2004. Intellectual property battles in a technological global economy: A just war analysis. *Business Ethics Quarterly* **14**(4), 679-693.

SCOZZI, B., CROWSTON, K., ESERYEL, U. Y., LI, Q. 2008. Shared Mental Models among Open Source Software Developers. *Hawai'i International Conference on System Science*. Big Island, Hawai'i. January 7-10, 2008.

SHAIKH, M., CORNFORD, T. 2003. Version Management Tools: CVS to BK in the Linux Kernel. **2005**.

STAMELOS, I., ANGELIS, L., OIKONOMOU, A., BLERIS, G. L. 2002. Code quality analysis in open source software development. *Information Systems Journal* **12**(1), 43-60.

STARK, J. 2002. Peer reviews as a quality management technique in open-source software development projects. *Software Quality - Ecsq 2002*. **2349,** 340-350.

STEWART, K. J., AMMETER, T. 2002. An exploratory study of factors influencing the level of vitality and popularity of open source projects. *Proceedings of the Twenty-Third International Conference on Information Systems*. Seattle, Washington, USA. December 14-17, 2003.

STEWART, K. J., GOSAIN, S. 2001. Impacts of ideology, trust, and communication on effectiveness in open source software development teams. *Twenty-Second International Conference on Information Systems*. New Orleans, Louisiana, USA. December 16-19, 2001.

STEWART, K. J., GOSAIN, S. 2005. The Impact of Ideology on Effectiveness in Open Source Software Development Teams (updated on 08/2005, opensource.mit.edu.

TUOMI, I. 2004. Evolution of the Linux credits file: Methodological challenges and reference data for Open Source resea. *First Monday* **9**(6).

VAN WENDEL DE JOODE, R. 2004. Managing Conflicts in Open Source Communities. *Electronic Markets* **14**(2), 104-113.

VAN WENDEL DE JOODE, R., DE BRUIJNE, M. 2006. The Organization of Open Source Communities: Towards a Framework to Analyze the Relationship between Openness and Reliability. *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*. Kauai, HI, USA. January 4-7, 2006.

VAN WENDEL DE JOODE, R., EGYEDI, T. M. 2005. Handling variety: the tension between adaptability and interoperability of open source software. *Computer Standards & Interfaces* **28**(1), 109-121.

VASS, B. 2007. Migrating to Open Source: Have No Fear. *3rd DoD Open Conference: Deployment of Open Technologies and Architectures within Military Systems*. Vienna, VA. December 11-12, 2007.

VEMURI, V. K., BERTONE, V. 2004. Will the Open Source Movement Survive a Litigious Society? *Electronic Markets* **14**(2), 114.

VERMA, S., JIN, L., NEGI, A. 2005. Open Source Adoption and Use: A Comparative Study Between Groups in the US and India. *Americas Conference on Information Systems (AMCIS 2005)*. Omaha, Nebraska, USA,. August 11-15, 2005.

VON HIPPEL, E. 2001. Innovation by user communities: Learning from open-source software. *Mit Sloan Management Review* **42**(4), 82-86.

VON HIPPEL, E., VON KROGH, G. 2003. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science* **14**(2), 209–213.

VON KROGH, G., SPAETH, S., HAEFLIGER, S. 2005. Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. Big Island, HI, USA. January 3-6, 2005.

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

WALLI, S., GYNN, D., ROTZ, B. V. 2005. The Growth of Open Source Software in Organizations. A report available at http://www.optaros.com/news/white-papers-reports#growth-open-source-software-organizations (retrieved on May 8, 2008).

WARING, T., MADDOCKS, P. 2005. Open Source Software implementation in teh UK public sector: Evidence from the field and implications for the future. *International Journal of Information Management* **25**(5), 411.

WATSON-MANHEIM, M. B., CHUDOBA, K. M., CROWSTON, K. 2002. Discontinuities and Continuities: a new way to understand virtual work. *Information, Technology & People* **15**(3), 191-209.

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

WEST, J., O'MAHONY, S. 2005. Contrasting Community Building in Sponsored and Community Founded Open Source Projects. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. Big Island, HI, USA. January 3-6, 2005.

WYNN, D. 2003. Organizational Structure of Open Source Projects: A Life Cycle Approach. *Proceedings of the 7 Annual Conference of the Southern Association for Information Systems (SAIS)*. Savannah, Georgia, USA. February 27-28, 2004.

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

YAN, N., LEIP, D., GUPTA, K. 2005. The use of open-source software in the IBM corporate portal. *IBM Systems Journal* **44**(2), 419.

YE, Y., KISHIDA, K. 2003. Toward an Understanding of the Motivation of Open Source Software Developers. *Proceedings of 2003 International Conference on Software Engineering (ICSE)*. Portland, Oregon, USA. May 3-10, 2003.

ZIMMERMANN, T., WEISSGERBER, P., DIEHL, S., ZELLER, A. 2005. Mining Version Histories to Guide Software Changes. *IEEE Transactions on Software Engineering* **31**(6), 429.