Collective Action and Communal Resources in Open Source Software Development: The Case of Freenet[1]

Georg von Krogh
Stefan Haefliger
Sebastian Spaeth
Institute of Management
University of St. Gallen, Switzerland
stefan.haefliger@unisg.ch
May 2003

We know that the postindustrial society is new because it triggers new forms of collective action

- Jean L. Cohen

Abstract

Building on resource mobilization theory, we explore three distinct rewards for individuals to engage in innovative collective action, namely open source software development. The three rewards, which we term communal resources, are reputation, control over technology, and learning opportunities. The collective action (the open source software development project) produces the communal resources in parallel with the actual product (software) and mobilizes programmers to spend time and effort, and contribute their knowledge to the project. Communal resources appear as a byproduct to the production process and represent a public good of second order. We show that they increase in value for individuals along with their involvement in the community. Empirical data from Freenet, an open source software project for peer-to-peer software, illustrates both the levels of involvement and the communal resources.

Keywords: technological innovation, collective action, resource mobilization theory, Open Source Software development

[1] We gratefully acknowledge the efforts of Eric von Hippel, Karim Lakhani, Christina Wyss, Marla Kameny and participants at various research seminars,

1. Introduction

Society has a vital interest in encouraging and rewarding technological innovation. Presently, the *private investment model* suggests innovation will be supported by private investment and that private returns can be appropriated from such investments (Demsetz, 1967). To encourage private investment in innovation, society grants innovators limited rights to the innovations they generate via intellectual property law mechanisms such as patents, copyrights, and trade secrets. These rights assist innovators in obtaining private returns from their innovation-related investments (Dam, 1995; Liebeskind, 1996). At the same time, the limited monopoly control that society grants to innovators under the private investment model and the private profits they reap represent a loss to society relative to the free and unrestricted use by all of the knowledge that the innovators have created. However, society elects to suffer this social loss in order to increase innovators' incentives to invest in the creation of new knowledge.

Open source software projects[i] are becoming a significant economic and social phenomenon that runs counter to this private investment model. Thousands of Free and Open Source software projects exist today. Sourceforge.net, a host of such projects, report more than 600.000 participants working on more than 60.000 projects. The number of users of the software ranges from few to many millions, and users include individuals, firms, and national governments. Well-known examples of open source software where many users benefit are the following; the Linux kernel of the GNU/Linux operating system, the GNU Network Object Model Environment (GNOME), which is software for a graphical desktop environment, the Perl programming language, Apache, the software for Internet servers, and the Internet e-mail engine SendMail. Counter to the private investment model, such projects produce software innovations that are public goods characterized by non-excludability and non-rivalry (Olson, 1965: 14). Open source software is usually made publicly accessible and one user's application of the software does not diminish any other person's utility derived from the software. Open source software is protected by licenses that do not secure innovator's profits, but rather the rights of users to download the code, investigate, modify, apply and redistribute the software for free. Thus, the developer both allows and promotes knowledge sharing through the legitimate distribution of the software. Therefore, open source software development projects avoid the social loss problem that is associated with the restricted access to knowledge of the private investment model (von Hippel and von Krogh, 2003). However,

they also create a significant puzzle, as stated by two economists Josh Lerner and Jean Tirole (2000), who have posed the following qestion regarding the open source software phenomenon: "*Why should thousands of … programmers contribute freely to the provision of a public good?"* As a public good, open source software does not yield the same opportunities developers could obtain by appropriating returns from their investment in a product protected by intellectual property rights. Also, as a public good, open source software are subject to the problem of free riding, where any potential beneficiary of the software has the option to hold back her own development efforts, waiting for someone else to produce it. Conventional wisdom suggests that for this beneficiary to contribute software code, sufficient incentives must be available encouraging contribution and punishing defection (Olson, 1965). Someone must bear the cost of incentives and as well mechanisms to control and safeguard the programming efforts of the beneficiary. However, free riding is also inauspicious to provisions of the public good (of first and second order): this is frequently referred to as the collective action dilemma in the literature (Oliver, 1993). In other words, why should open source software development projects exist at all?

For several decades, resource mobilization theory has studied the conditions necessary for collective action to happen in society. In particular, authors have been interested in characteristics of organizations that provide members with sufficient rewards to act collectively (e.g. McCarthy and Zald, 1977). In this paper, against the backdrop of resource mobilization theory, we pose the following question: *What are the sufficient conditions that mobilize programmers to contribute freely to the provision of a public good?* Based on an exploratory case study of Freenet, a project developing sophisticated software for peer-to-peer file sharing over the Internet, we show that in the collective action of producing open source software, the project *in parallel produces a set of communal resources*. These resources represent a public good of the second order, and resolve the collective action dilemma referred to earlier. Communal resources are collectively produced by the project, and provide individual rewards for developers, and these rewards increase with the developers' involvement in the open source project. The paper is organized as follows: Section 2 briefly introduces the open source software phenomenon for the benefit of the unfamiliar reader. Section 3 discusses the theory we use to generate the research question and interpret the case data. Section 4 provides an overview of our research methods, and Section 5 a description of the case. Section 6 demonstrates communal resources in Freenet, and Section 7 concludes the paper by discussing implications for theory and research.

2. The open source software phenomenon

Open source software development is radically different from software production in the private-investment model. The first and obvious contrast is the work of *hackers*[ii] and their communal practices of sharing software code. Open source software development projects were first recorded among communities of hackers, and particularly a group of programmers housed at the MIT's Artificial Intelligence Laboratory in the 1960s and 1970s (Levy, 1984). Programmers routinely exchanged software with each other and built upon each other's software both individually and collaboratively. In such hacker communities the sharing of software, collaborative learning, and openness became part of everyday life. Rarely did people question the economic rationale of their practices (Levy, 1984). With the emergence of the Internet, the communal practices among hackers began to spread rapidly throughout the world (Himanen, 2001; Tuomi, 2002).

In the 1980s software development became increasingly commercialized (Cusumano, 1992). The incentive to innovate for many firms in the software industry, such as Microsoft, Sun Microsystems, and SAP, was safeguarded through efficient regimes of intellectual property protection. Innovators began to protect their software source code[iii]. A technical protection mechanism for proprietary software was to hide the human readable source code from the user so that he or she could not study or modify it (e.g. Moerke, 2000). Only the user interface, such as a word-processor, as well as the machine code became available to the user. Moreover, innovators protected proprietary software by copyright and patents (Dalle and Jullien, 2003; Simon, 1996). A software license allows an individual or group to use a piece of software for a certain purpose and/or period of time. Often this right to use is traded against a royalty paid to the producer of the software. A commercial firm, finally, may have software patents that cover particular innovative features of software often in conjunction with a business process, such as an algorithm for searching through very large amounts of data in a customer database (Mykytyn, Mykytyn, Bordoloi, McKinney, & Bandyopadhyay, 2002).

In the early 1980's a number of hackers led by Richard Stallman, a highly experienced programmer at MIT's Artificial Intelligence Laboratory, were especially concerned by the gradual loss of access to collectively developed source code. They also viewed with suspicion the general trend towards the creation of proprietary software packages and the release of software in forms that could not be studied or modified by everyone (Moody, 2001). Stallman's pioneering idea was to use the existing mechanism of copyright law to create a

new form of license, whereas a software author uses his or her own copyright license or affixes any of a number of standard licenses to the code in order to guarantee a number of rights to all future users, a technique called *copyleft*[iv]. Typically, these rights ensure that a user possessing a copy of an Open Source software program has the legal right to use it, to study the software's source code, to modify the software, and to distribute modified or unmodified versions of it to others.

The hacker practices of software development also differ radically from commercial software projects. Open Source software projects are initiated by an entrepreneur and rely on voluntary, collective participation of individual developers from technical communities or organizations who contribute by virtue of solving their specific technical problems. The developers are rarely paid for their services and normally there is no contract governing the relationship between the project and the developer (Tuomi, 2002). In contrast, for proprietary software projects, the entrepreneur uses the two mechanisms of protection for rent appropriation. These are most effectively executed in the firm (Liebeskind, 1996), because it secures the continued collaborative efforts of other developers, minimizes free-riding and opportunism among developers, as well as secure firm-specific learning and cash flow needed for developing future product generations (e.g. Conner and Prahalad, 1996; Meyer and Lopez, 1995; Young, Smith, & Grimm, 1996). In proprietary software development the relationship between the firm and developers is therefore regulated by contract and monitored for compliance (Austin, 2001).

In sum, the combination of a regime of protection of user's rights to the software and the specific hacker practices of voluntarily developing software is surprising and begs the question of what conditions mobilize programmers to contribute freely to the provision of a public good. In the next section we turn to resource mobilization theory for possible answers to this question.

3. Theory

There exist many legitimate and important ways of studying collective action in the Open Source software movement, ranging from the new social movement theories' analysis of political and cultural aspects of the hacker communities (see Buechler, 1995 for a review), to rational choice theories' analysis of cost and benefits of individual developers' participation (see Ostrom, 1998 for a review and extension). However, our interest in this paper was

generated mainly from Lerner and Tirole's observations of the "irrational", voluntary aspects of the public good production, and the intriguing puzzle they formulate. Beyond the individual interest and motivations of developers to contribute to the public good, Open source software projects, such as the Linux project (Moon and Sproull, 2000) and the Apache project (Lakhani and von Hippel, 2003), are also social movements that mobilize resources for this production, and within a rational choice and instrumentalist framework of analysis these projects derive their own status and deserve further examination. Hence, our research question can be considered a reformulation of Lerner and Tirole's question: What are the sufficient conditions that mobilize programmers to contribute freely to the provision of a public good?

Developed in opposition to a Durkeheimian view of collective action, which is described as the result of dark irrational passions of movement members and the breakdown of society's normative control over individuals, resource mobilization theory emphasized the instrumental motivations of groups forming social movements. Whereas breakdown theories capture the unrest and mal-integration behind non-routine collective action such as riots and rebellion, resource mobilization theory best explains routine collective action such as rallies and protests (Piven and Cloward, 1992; Useem, 1998). The resource mobilization perspective argues that organization underlines successful collective action projects (Tilly, 1978), including the professionalization of such organizations, the career patterns of social movement personnel, and the emerging social movement industries (McAdam, McCarthy, & Zald, 1988). According to a review by Baron and Hannan (1994), until the synthesizing essay by McCarthy and Zald (1977), the contributions to resource mobilization had been rather scattered and transient, competing for attention with studies that emphasized grievances, frustration, beliefs and collectivity of individual actors (see also McCarthy and Zald, 1973). The elegant solution for explaining collective action proposed by McCarthy and Zald (1977) was to look beyond theories and empirical work on the social psychology of grievance. In their project, the motives individual contributors might or might not have for contributing to the public good succumb to the relative importance of the means by which collective action is organized (McCarthy and Zald, 1977; see Oliver, 1993 for an extensive review). In the instrumentalist view of collective action and the resource mobilization framework of analysis, the creation and deployment of selective incentives for contributors to allocate sufficient resources is essential to the success of collective action projects (e.g. Friedman and McAdam, 1992; Oliver, 1980)[v]. However, selective incentives, as Mancur Olson (1965) postulated

them, cannot explain collective action, since selective incentives represent a public good as well and somebody needs to pay for them in the public's interest (Oliver, 1980; 1993).

Open source software development can be characterized as routine collective action, as it is non-violent and not aimed at challenging established order or overturning normative control in society. However, open source software development deviates from collective action usually analyzed in the resource mobilization theory in four ways, which shed new light on the role of selective incentives. First, knowledge is both a resource for the project and its goal, second, the development process represents the central activity over a long period of time, third, goal directed recruiting is largely absent, and finally, no measures are taken to prevent free-riding on the public good. First, McCarthy and Zald's (1977) concept of the resources to be aggregated by social movement organizations covers money and labor (p. 1216), as well as time (p. 1227). In their reasoning, social movement industries thrive under conditions of resource abundance in society, provided that the general level of disposable income raises, as does the shift in control over work schedules from upper echelons of managers to lower level employees in firms. In their work, they put no additional constraint on the type of resources the organization need in order to survive (p. 1226). In order to meaningfully conduct an analysis of the Open Source software phenomenon, an additional resource must be added to this framework: knowledge. It is a well-known fact among writers in the resource mobilization tradition that social movements create knowledge as a by-product of its activities, and thereby generate considerable value for society (e.g. Eyerman and Jamison, 1991; Flora and Flora, 1993; Herman, Wolfson, & Forster, 1993; Indyk and Rier, 1993; Myers, 1994)[vi]. However, resource mobilization theory thus far did not consider collective action that has the *production* of knowledge as the primary organization goal, which is the case for an open source software project (software code) (see von Hippel and von Krogh, 2003). For example, a project that develops software for extreme protection of privacy when sharing information over the Internet needs programmers who can bring in specialized knowledge in cryptography. This knowledge is probably a rare resource held by only few firms, research institutions, universities, or individuals. Therefore, an additional constraint is introduced in the analysis of open source software projects. Not only do these compete for time, money, or labor in the social movement industry and sector, provided they have the production of knowledge and innovation as their ultimate goal, but also they engage in intense competition for the best knowledge and the most talented. De facto, many contributors to Open Source software projects have regular jobs at commercial software firms (Gosh, Glogg,

Krieger, & Robles, 2002), and they either spend part of their work or leisure time on developing code for these projects. Moreover, if knowledge is a resource on which projects compete, not all contributions are likely to further the project's goal of innovating[vii]. As a consequence, the development process assumes central importance within collective action. Certain projects (movements), such as Linux, can exist for ten years and more without the full functionality of the software being reached. When the goal of the collective action is the production of knowledge, the way to reach this goal is part of the goal, and the role and character of selective incentives may change.

In the resource mobilization theory individuals are seen as rational actors who engage in instrumental actions and who use organizations to secure resources and foster mobilization (McCarthy and Zald, 1977). The resource mobilization literature has placed a great deal of emphasis on the importance of incentives for joining a movement, cost reduction mechanisms for making contributions, and career benefits of such behavior (McCarthy and Zald, 1977; and their emphasis on Oberschall, 1973). Recruiting and properly motivating participants in a successful collective action project, in order to increase the attractiveness of contributing, assumes central importance. With respect to successfully recruiting contributors to a collective action task, especially where information about the movement is scarcely distributed among potential participants, many writers predict that both the specification of project goals and the nature of recruiting efforts should matter a great deal (Benford, 1993; McPhail and Miller, 1973; Snow and Benford, 1992; Snow, Zurcher, & Ekland-Olson, 1980). Thus, direct and stable social relationships between recruiters and potential participants are important, so that recruiters will have more information about individual motivations and thus be more effective in defining a rewarding goal (Oliver and Marwell, 1988; Taylor and Singleton, 1993)[viii].

Because open source projects need knowledgeable developers one would expect that they would engage in precise goals formulation and active recruiting. However, successful Open Source software projects do not appear to follow any of the guidelines for successful collective action projects just described. With respect to project recruitment, goal statements provided by successful Open Source software projects vary from technical and narrow to ideological and broad – and from precise to vague and emergent (for examples see goal statements posted by projects hosted on Sourceforge.net)[ix]. Further, such projects typically engage in *no* active recruiting beyond simply posting their intended goals and access address

on a general public website customarily used for this purpose (for examples, see the website named “Freshmeat.net”). However, potential participants can access information about the project by performing a search on the Internet by such tools as Google. Beyond communication over the Internet, people rarely meet face-to-face. Some people participate in Open Source software development projects under pseudonyms concealing their real identity, thereby obscuring their interests to the entrepreneur (see Sections 5 and 6). Even under these seemingly adverse conditions, projects such as Linux or Apache have shown that they can be successful in attracting large groups – perhaps thousands - of contributors.

Finally it is interesting to note, open source software projects seem to expend no directed effort to encourage one to contribute rather than be a free rider. Anyone is free to download code or to seek help from project websites, and no apparent form of moral pressure is applied to make a compensating contribution (Lakhani and von Hippel, 2003). Deviating from existing theory, what can explain this type of collective action? A lead can be found in John Elster's (1986) work. He observed that the instrumentalist ideas of collective action embedded in such theories as resource mobilization did not put enough emphasis on the rewards ensuing from process related aspects of collective action. Elster remarks (1986: 132) :

> "*..cooperation reflects a transformation of individual psychology so as to include the feeling of solidarity, altruism, fairness, and the like. Collective action ceases to become a prisoner’s dilemma because members cease to regard participation as costly: It becomes a benefit in itself, over and above the public good it is intended to produce”*.

Recent developments in economic theory support Elster’s conjecture. Thus, Rabin (1993) and Fehr and Schmidt (2000) have shown that a game, which in material payoffs constitutes a Prisoner’s Dilemma, can be transformed into a coordination game in which cooperation is also an equilibrium outcome if pecuniary motivations *and* social motivations are taken into account. Beyond theoretical and laboratory-based work, Elster's conjecture did not receive much attention in field studies of resource mobilization. However, his work has important ramifications for empirical studies: individuals are expected to join collective action for rewards, other than that promised by the movement's end goal. In other words, there should be *private* rewards apart from and before the end goal to those that contribute to Open Source software projects, which should be considerably stronger than those available to free riders[x]. This implies that the development process may be as important as the (envisioned)

final product, which cannot be supplied at once (e.g. GNU/Linux has evolved over a period of 12 years!). Programmers are rewarded for their contributions, and do not appear to wait.

In line with Elster's conjecture, we propose that *the production process of knowledge in an open source software project has as a byproduct communal resources that reward its contributors*. The empirical phenomenon of Open Source software development represents a form of collective action where the second order public good (Olson's selective incentives, or the communal resources in our case) emerges from the production process of the original public good (here: software). In this constellation the "collective action dilemma" that Pamela Oliver (Oliver, 1993: 274) ascribed to Olson's work disappears. The characteristics of and the accessibility to the communal resources are the sufficient conditions in our reformulation of Lerner and Tirole's research question. Based on our proposition, we shall empirically explore these communal resources in the case of Freenet (Section 6), but first we turn to the research method (Section 4) and then introduce the case (Section 5).

4. Design and method

The purpose of our exploratory case study design (Yin, 1994: 30)(Yin, 1994:30) is to investigate communal resources built during the development process, rewarding contributors to an open source software project. The case study allows us to explore our proposition (Hartley, 1995; McPhee, 1995) on the sufficient conditions for routine collective action to happen in the novel setting of open source software development. The research had three phases: sampling of the case, data gathering and building a case study data base, and data analysis. We draw on both qualitative and quantitative data as both complement the insights into the functioning of the Freenet project (Silverman, 1993).

The case was sampled for three reasons. First, in contrast to Linux, Freenet was launched not on the basis of workable code the entrepreneur had written and revealed (Lerner and Tirole, 2000), but rather on a master thesis in computer science published on the Internet by its founder Ian Clarke, outlining the theoretical principles of anonymous peer-to-peer computing (Clarke, 1999). This gives weight to an ambitious goal of creating new knowledge rather than improving already existing software. As will be seen, the development of Freenet hinges on knowledge of cryptography, which is considered a rare resource among experts on software development. Second, Freenet has no template of software architecture available, such as Unix for Linux (Wayner, 2000), making Freenet a radical innovation of peer-to-peer

software[xi] (Boehm, 2000; Oram, 2000). In effect, there is no example of modularization (easy identifiable clusters of files and functions) of the software available. Hence, we reasoned that those involved in the project would not have an easy understanding and access to the technology. In contrast to projects emulating existing applications. Freenet contributors do not know in advance what to expect of their final product. Therefore, the cost of contributing should be considerably higher than for projects where templates and architectures are available (see Waterson, Clegg, & Axtell, 1997). Third, Freenet is young in comparison to Linux for example, which has been in operation since 1991. This gives Freenet a ”liability of newness": The project must compete for knowledgeable developers with other established projects that have routines available for resource mobilization. This might decrease the likelihood of the project's survival compared to that of existing projects (McCarthy and Zald, 1977). Our data covers the calendar year of 2000. This first year was particularly important for collective action because it involved the mobilization of 26 developers to a total of 30. Given a fairly stable group of developers, we reasoned that 2001-2003 did not provide much additional insight into the sufficient conditions for mobilization of developers. These first three characteristics make Freenet in its first year of existence, a critical pilot case with respect to studying our proposition (Glaser and Strauss, 1967; Stake, 1995; Strauss and Corbin, 1990; Yin, 1994).

We gathered data from four different sources. First we conducted thirteen *telephone interviews* in three rounds with Freenet core developers identified from the developer list on the project’s homepage. Each interview took between one and two hours and was recorded and transcribed to facilitate easy data analysis. The rounds occurred between October, 2000 to January, 2001 and March to May, 2001. All interviews were semi-structured with guidelines including developer background information, overall structure of the project, reason for joining and working on the project, rewards, specialization, and particular challenges in the project. In May 2003 we emailed an electronic questionnaire to participants in the project in order to obtain further insights on communal resources.

Second, we collected the contributors’ public email conversations stored in the projects’ *mailing lists,* which are archived on the publicly available website Geocrawler[xii]. Since we focused on the contribution to technical development of Freenet, we gathered e-mail data from the ‘development’ list where contributors discuss themes pertaining to the next release of Freenet on the Internet, its design, emerging architecture, and other technical

aspects. A challenge in case studies is organizing large amount of data for effective analysis (Yin, 1994). We created a case study database organized on month-by-month basis, of messages including contributor identity, date and time for posting a message, mails responding to the message, and mail content for the period January 1 – December 26, 2000. The database included approximately 12,000 single email messages from 356 unique participants (von Hippel, von Krogh, Lakhani, & Spaeth, 2002). However, no attempt was made to extend the analysis to measure the number of lurkers on this list[xiii].

The third source of data included committed source code within the *CVS ('Concurrent Versions System').* CVS is a public 'version control' tool, designed to synchronize work and keep track of changes in the source code performed by developers working on the same file. CVS stores its version-control information in a directory hierarchy on a central server, called 'the Repository', separate from the user's working directory. The repository allows developers to add or remove files easily, or to ask for information about a set of files. The CVS also stores the developer's comments that document their work. While the source code and comments are publicly revealed, in Freenet only a restricted number of about 30 core developers can commit source code to CVS. In the case study, source code commits served as a major pool of data because progress of the project is reflected by the progress of source code modifications. The data we retrieved from the period January 1 – December 26, 2000 included 1244 source code commits from the 30 developers. This comprised in total 54.000 lines of software code added, excluding the initial revisions of code.

Fourth, we collected and analyzed *publicly available documents* related to Open Source in general and to the project in particular, including the Freenet project web pages (e.g. the Frequently Asked Questions (FAQ)[xiv]), Ian Clarke's master thesis (1999), newspaper articles on peer-to-peer software and interviews with the core developers[xv], a working paper describing the Freenet technology (Clarke, Sandberg, Wiley, & Hong, 2000), and a presentation on results of a simulation of the software (Hong, 2001). Where there may have been doubt, ambiguity, or lack of data, clarification with individuals in the project was obtained via e-mail.

In the sections that follow, we describe Freenet (Section 5), and the results of our data analysis (Section 6).

5. Freenet

Freenet is a peer-to-peer network designed to allow the distribution of information over the Internet in an efficient and anonymous manner. Ian Clarke started the Freenet project when he was a fourth year computer science student at the University of Edinburgh, and completed the basic design in 1999. The overall goals of Freenet as defined by Ian Clarke are (freenet.sourceforge.net, 2000):

- The network should have no centralized control or administration
- It should be virtually impossible to forcibly remove a piece of information from the network
- Both authors and readers of information should remain anonymous if they wish to do so
- Information will be distributed throughout the network in such a way that it is difficult to determine where the information is being stored
- Availability of information should increase in proportion to the demand for that information thus preventing the *Slashdot effect*[xvi]
- Information moves from parts of the Internet where it is in low demand to areas where demand is greater

These first basic ideas and design were outlined in his master thesis entitled "*A distributed decentralized information storage and retrieval system*", which was published on the web for people to comment on, in particular on how to turn these ideas and design into a workable software. The original document received limited attention, and Ian sent an email to the subscribers on the mailing list around Christmas 1999 announcing that he planned to step down as project leader due to personal reasons. Nobody volunteered to take over his position, but a number of people on the mailing list came to realize that there was a minimal amount of programming going on at the time. This spurred a revitalization of efforts, and in early 2000 several people made an active contribution to Freenet.

The first release of Freenet on the Internet happened in April 2000. Nine releases of Freenet were made in 2000 and eight releases in 2001. Since then, the Freenet software has been downloaded more than 1.400.000 times from the Internet. This shows the significant

public interest in this software, which also possibly triggered the interest of many potential new developers to the project.

Collective action in Freenet includes the discussions on the development mailing list that accompanies the production of the software code. Figure 1 shows the overall community size over the year and the number of people joining and leaving the list. The date people join the list is derived from the date of the first mail a person sent to the development list, while leaving dates are derived from the last mail sent to the list.

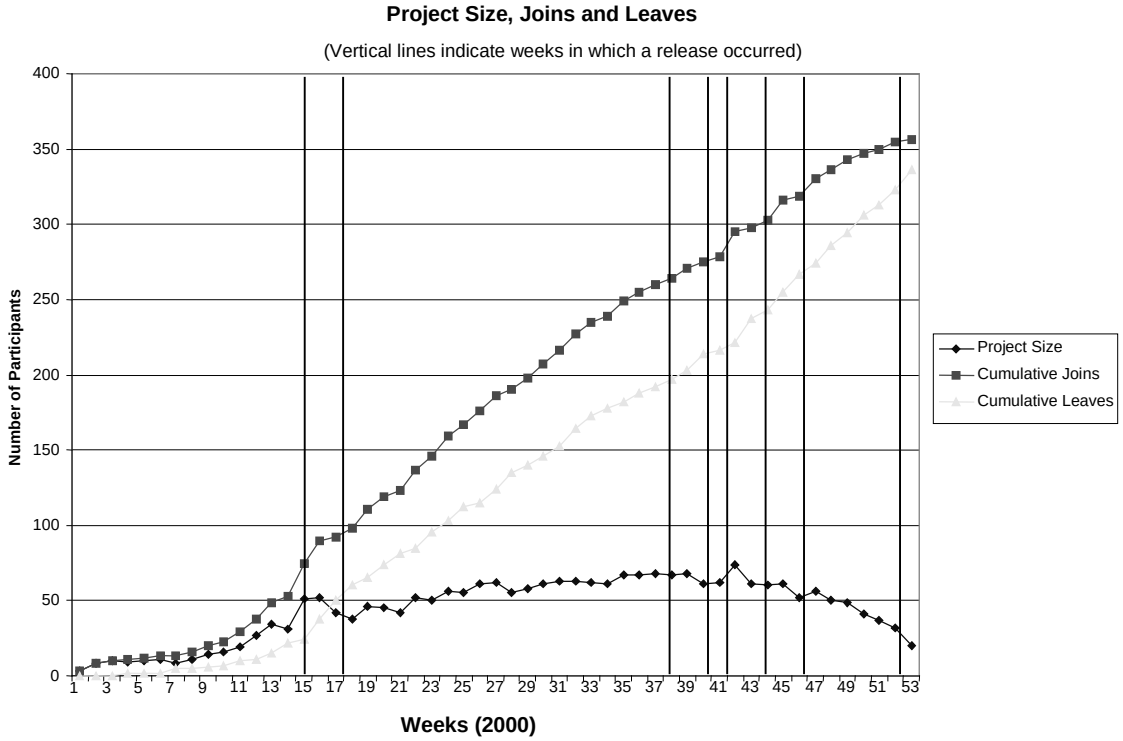

Figure 1: Project size based on e-mail activity (based on von Hippel et al., 2002)

On average the Freenet project consisted of 45 (sd:21) active participants per week. This number was achieved around the first public release date of the project and after that remained fairly stable. In 2000, 356 individuals participated in the main Freenet developer discussion list. They generated 11,210 e-mail messages over 1,714 message threads. A message thread is given by responses to an initial e-mail, covering a specific issue.

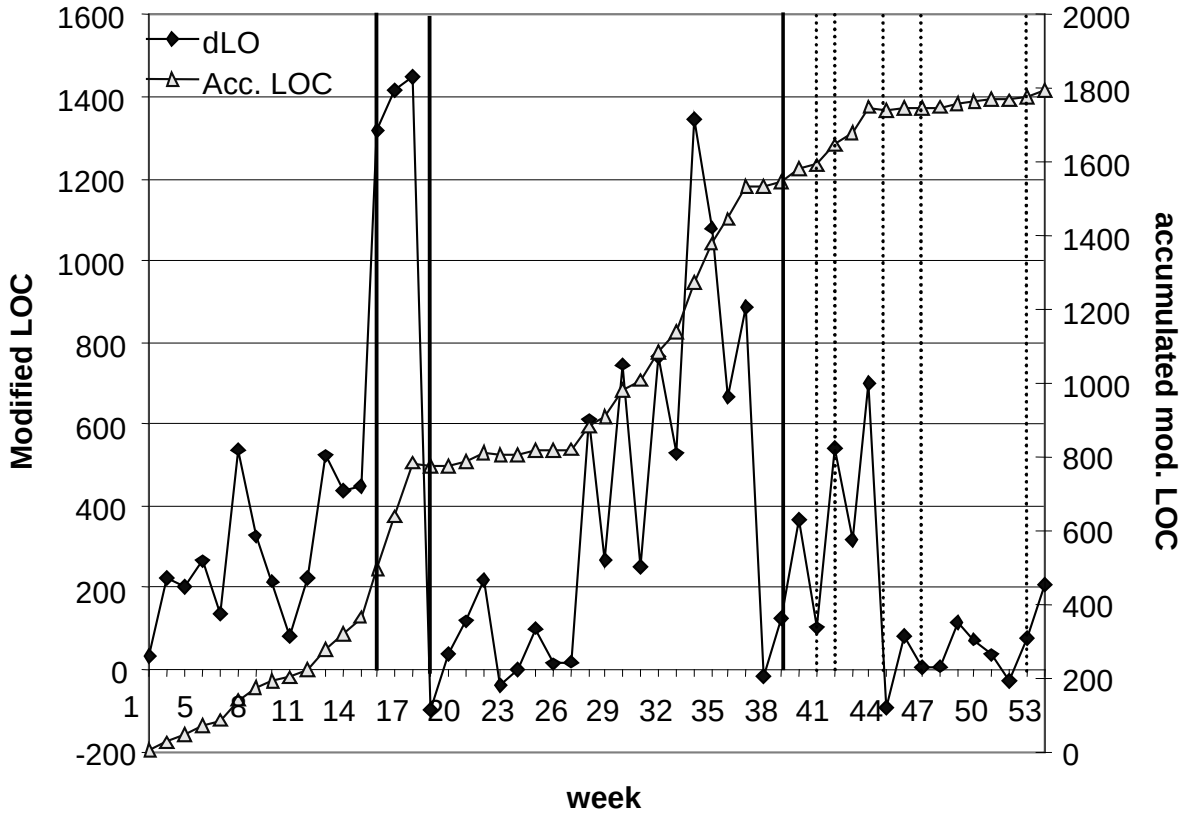

Figure 2, Added LOC and accumulated added LOC

Figure 2 visualizes the number of modified lines of code (LOC) in the source code repository. The left scale shows the number of added lines of code in each week (negative values mean more lines have been removed than added), the right scale displays the accumulated number over time. It is to note, that this measure captures only changed LOC in already existing files. The first revision of a file which is imported into the CVS system does not count as modified and its LOC are not counted. Therefore the accumulated added LOC do not exactly correspond to the total number of lines of code which exist in the source code repository. When the first public release of Freenet (0.1 Beta) was made in week 15 (in 2000), this resulted in a steep increase in source code changes, and sparked the interest and entry of potential new developers into the project. Although Freenet was not started based on working code, this shows that having working code in a project can indeed mobilize new developers as suggested by the open source software expert Eric Raymond (see Raymond, 1999). In the next section we proceed with an analysis of the case data regarding the rewards that ensue for those who contribute this substantial amount of development work to Freenet.

6. Communal resources

Consistent with resource mobilization theory, it is only possible to mobilize hackers to contribute to an Open Source project if the individual cost-benefit analysis reveals outweighing benefits. We propose the characteristics of and access to three distinct rewards, or communal resources. Communal resources do not exist prior to the collective action and they are not used in a recruitment process or somehow targeted to the individual., Rather, they are produced during the development process and are accessible under certain conditions to self-allocating, individual programmers. The communal resources share along with Mancur Olson's concept of selective incentives (1965) the function of mobilizing individuals, where both represent the benefit-side of the individual rationale for participating in a collective action project. However, opposed to the selective incentives, communal resources are not provided by a particular person or institution that carries the cost (Oliver, 1993). In our case study this problem does not arise, since the communal resources emerge from the production process of the Open Source software. A programmer in Freenet chooses, or aspires to a particular level of involvement in the project. Involvement, in turn, is linked to the extent to which the communal resource is accessible. In other words, the benefit that can be reaped from access to the communal resource increases with involvement.

In a first round of analysis, we searched for themes in the interviews that would reveal specific communal resources for the developers. We were looking broadly for statements touching upon what people experienced as developers in Freenet and why they contributed to the project. This form of analysis is very similar to what Barley called "systematization" of topics emerging from his observations of radiology departments (Barley, 1990). We found topics that could be grouped in three communal resources: reputation, control over technology and learning opportunities. Reputation and control over technology in a collective development effort cannot exist before the community does, and learning opportunities open with the observation of work or the interaction of the programmers. All three communal resources are features of the development process[xvii]. In subsequent iterations of analysis we sought to make the three constructs operational.

Involvement

For the analysis of Freenet we operationalized involvement into three categories: lurkers (the lowest level of involvement), contributors, and developers. Lurkers eavesdrop on

mailing or discussion lists without posting (Nonnecke and Preece, 2000). This group helps to promote standards by using the software, to spread reputation, and represents a pool of potential new developers, since nearly all developers started out by simply reading the mailing lists and trying out the software (von Krogh, Spaeth, & Lakhani, 2003). As mentioned in Section 4, we could collect no data on lurkers, although some interviewees revealed they had lurked for an extended period of time before becoming developers.

The second group, the (regular) contributors, constitutes the largest visible group of affiliates to the project. They either contribute code via a "gatekeeper" (developer), who has CVS write access and evaluates the submission, or they participate in the discussions. The breadth of involvement for contributors is large. At one extreme, a contributor may demand an exorbitant feature (few might even value a contribution being a mere request), where at the sophisticated end a contributor may be involved in technical discussions and regularly submit useful patches.

Among the 30 developers we found several hackers who were particularly active: Oskar Sandberg, Scott Miller and Brandon Wiley. Involvement is generally associated with increased technical skills, merit and a sense of responsibility (Clarke, 2000). The developers take on the vast majority of coding, plan the version releases, and decide on the inclusion of features or any issues determining the overall direction of the development. Among them, the founder of the project, Ian Clarke, has a right to veto suggested technical changes. Technically speaking, the developers consist of all the participants with CVS write access. The CVS hosts the most current version of the program and allows multiple programmers to simultaneously change code. Thus, CVS write access is synonymous with a position to change the official code version. Technically, anyone could download and recreate this system and build a second competing version of Freenet, but in practice the authority of the official version is rarely questioned (Wayner, 2000). Access is granted to a relatively small circle of skilled programmers who have earned the trust of other developers through their contributions and effort given to the project (von Krogh et al., 2003).

In the following, we will argue that the level of involvement influences the individual access to the communal resources. Table 1 synthesizes the types of rewards corresponding to the level of involvement and the communal resources.

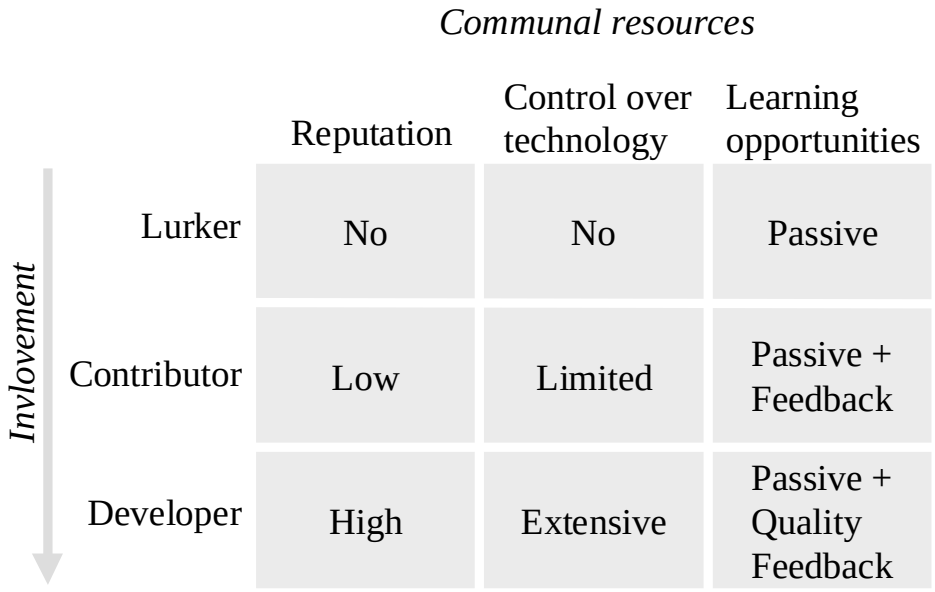

Table 1: Summary of the findings for involvement and communal resources

Reputation

Reputation is attributed to a person and develops over time. It reflects other observers' judgment and does not imply a value statement. Reputation can be positive or negative and it can improve or deteriorate. We limit our analysis of the data to the establishment of a good reputation in Freenet. Of course, there exist reputation losses and various forms of sanctions in Open Source communities that are not considered here (for more on this, see O'Mahony, 2003). Reputation can only exist vis-à-vis an audience. Since Open Source software development projects consist of experts in computer programming, and the daily work on coding and the technical discussions for a particular project hardly ever resonates outside the project, it makes sense to distinguish the audience inside from the one outside the project. Thus, reputation is created within an Open Source project and may under certain conditions, spill over to a wider public on the outside. We distinguish the audiences outside and inside the project and present reputation within the project as a communal resource by showing a relationship to the level of involvement in the project.

Using a labor market argument, Lerner and Tirole (2000) and Lee et al. (2003) suggest developers who build a strong reputation in the Open Source projects would enhance their human capital, and so would raise their value to a future employer. An intriguing idea, the interviews we conducted with the developers of Freenet did not confirm that they contributed

in order to raise their value in the labor market for software engineers. Such as a developer who calls himself "Mr. Bad", even used fictitious identities in order to conceal their real identities and often their affiliations with software firms. Rather, it seems the direct effect of reputation among peer Open Source software developers was important to Freenet developers, which is consistent with assertions in the literature (Himanen, 2001; Lerner and Tirole, 2000; Raymond, 1999; Risan, 2001). As core-developer Scott Miller put it:

> *"If I am writing a program why not release it as Open Source. It doesn't cost me anything and, you know, it might be good for my reputation, it might get me involved in other things. And I think that's how I got started programming in it."*

Or as Oskar Sandberg, also a developer, admitted*:*

> *"Yeah, in a way. I like to, I like to do something extra, I try to make my code nice or make the other people I work with like it (laughs). And yeah, of course it's always good to hear nice things about your work and such things. It would be harder to work if I wouldn't get any further positive feedback, of course. (...)*"

However, the global hacker community celebrates its own stars such as Eric Raymond, Bruce Perens, Ian Clarke, and Linus Thorvalds. These are people who have made major contributions in terms of software code, and hence based on their merits attained a strong reputation. Even further away from the hacker culture, there exist examples of hackers who attracted a wider public attention through media coverage such as books for the non-hacker (e.g. Moody, 2001), a foremost example Linus Torvalds (Torvalds and Diamond, 2001). As these examples are very rare, worldwide fame cannot be expected from an involvement in an Open Source community. What can be expected are like-minded hackers and experts in their fields of programming who scrutinize new code and evaluate its functioning, and consequently the achievement of its author. Similar to written text, the criteria for evaluation of software code seem unlimited and hinge on personal preferences ranging from viewing code as a simple tool to a form of art (e.g. Knuth, 1969).

In a technical community that judges people on merits rather than personality, a programmer's reputation is mainly built over time through excellence in coding (Kohanski, 2000; Raymond, 1999). In Freenet, although code patches[xviii] were sent to core-developers by various contributors (prior to them receiving CVS access or as a singular contribution), the core-developer had what we would call a "fingerprint" advantage. The code was committed to

the CVS identifying the developer who entered it into the CVS. This person was not necessarily, although was most of the time, identical to the author. At this point the code was open to scrutiny from the public. Our interviews revealed that the hackers typically found code (of others and themselves) to be "elegant" if it could perform a complex or difficult task yet required limited computer processing. If the code was small in terms of lines of code, but still important for the functionality of the overall Freenet software (see Kohanski, 2000), the core-developer had a unique possibility to build a reputation within the Freenet community among the other project participants, and potentially also outside the Freenet project environment. In Freenet, this type of reputation building was open only to the developers with CVS writing access.

Another way to contribute was to participate in development discussions. The mailing lists were public and anyone could read and comment. This form of contribution to the project did not directly require coding skills, though for competent participation (incompetent comments tend to be ignored), a certain level of technical expertise and sound knowledge of the state of the project was indispensable. Useful comments included references to various sources of information or new ideas on a problem discussed. As valuable as these contributions may be, they were "only" written speech and not source code.

We measured reputation as the number of neutral references and explicit praise given to a person within the community. A neutral reference to a person means the mention of the person's name in the mail discussion. The vast majority of references involved neutral or positive assessment of the person's work. We counted all references to core-developers in the developer mailing list of Freenet for the year 2000, except those with frequent names (where potential confusion compromised the quality of the analysis) and excluded four outliers (Ian Clarke, Oskar Sandberg, Brandon Wiley and Scott Miller) who each wrote on average three times more mails than the next frequent discussant. The 19 core-developers were referenced on average 93.9 times, with the median being 80 and the standard deviation a high 87.5. A randomly chosen sample of equal size out of the 356 contributors without CVS access revealed an average of 3.4 references with the median being 0 and the standard deviation 8.3. These differences in references reflect the number of mails sent to the list and hence reflect how well known core-developers were compared to regular contributors. On average, a core-developer sent 132 mails to the mailing list, whereas the average from our contributor sample was 11.2. In the analysis on explicit statements of praise, only 1 out of 20 lauds were

addressed to a regular contributor, whereas the other 19 were directed to core-developers. For example, in August 2000 Oskar Sandberg wrote:

> "*There are two separate projects to write Freenet nodes in C and C++. One is getting along only because of Adam's noble and excellent work, …*"

Or on a more humorous note a contributor by the name of Flute Gardener wrote in October 2000:

> "*Oskar [Sandberg] is better than god and eminem. Neither god nor eminem write Freenet code! Although it would be kinda cool if great code would inexplicably appear in the CVS.*"

Clearly, reputation was only accessible for the developers, who were mentioned regularly. The contributors were referenced far less frequently. Within the large group of contributors the number of mails sent to the discussion list ranged from 1 to 128. A Wilcoxon-Mann-Whitney rank sum test rejected with 99% confidence that the references of both samples stem from an identical distribution. Lurkers cannot have a reputation, for they never appeared publicly. Reputation is not automatic and can also be understood as a reputation opportunity, in this context. This analysis does not explain the causes of reputation, but the association suggests that reputation increases for the same reasons as involvement increases.

Control over technology

The second communal resource we identified was control over technology. Developers are able to control what and how the software is going to work, the compatibility or lack of compatibility with other software, and to a certain extent who were able to use the software through various means. As mentioned only 30 developers could commit the code that eventually comprises the software innovation. Furthermore, code developed by non-developers was reviewed and a developer decided on its potential inclusion in the source code repository. By submitting the source code themselves, the developers maintained a tight control over the source code and therefore the Freenet technology in terms of functionality[xix].

We found, for example, that some developers had strong ideological interests associated with Freenet, particular regarding anonymity. An important technical discussion continued among developers and contributors for weeks regarding the search for files in

Freenet. A file search function that would made the location of information easier for Freenet users, could at the same compromise anonymity of the sender and receiver of information and was deemed against the Freenet design goals (see Section 5) by the developers. While this discussion generated considerable discussion volume, the developers decided not to include any software files that would compromise the anonymity of the users. In this sense, a privileged access to the control of technology allowed developers better opportunities to realize their interests than non-developers in the project.

If *anybody* could impact on the technology developed, it seems likely that the utility people derive from belonging to the developer group would diminish rapidly. Consider that a malevolent developer would intentionally commit code to the CVS that cause the software to malfunction. If this were the case, the returns on the developers' personal investment in coordination, code writing, discussion, code reviewing, debugging and so forth would diminish rapidly. Hence, control over technology afforded to developers whose work one judges as valuable, secures all developers' future rewards associated with the development of the technology. Of course, this communal resource would not hinder an existing developer in committing code that would break down the technology. However, in Freenet, for a contributor to become a developer with CVS access required on average 23.4 messages before he was given access. These messages contained highly technical content such as suggestions to fix a bug in the software, a code patch, or a software-related commentary or review. This relatively lengthy and costly process of joining the project made it easier for the existing developers to identify a capable new developer, and any adverse action, once privileged access had been given, would have caused a negative reputation beyond the project within the hacker community at large (Raymond, 1999).

Being able to control what functionality was worked on in order to satisfy one's own personal needs proved to be a communal resource in order to mobilize contributions to the Freenet project. This interview quote from a developer illustrates this point:

> "*My main interest, at the time, was getting my magazine published on Freenet. There weren't really tools for doing a full Web site, and I spent some time working on making a tool for that, working on fixing the client interfaces, and in some marginal areas of the [core parts], to make it work, so my magazine could get published.*" (Mr. Bad)

The quote also shows that developers controlled what was being worked on in the technology due to the self-appointment of tasks. This seems typical for open source projects (Schoonhoven, 2003), for as we argued in Section 2, unlike in a software firm, contributors and developers in such projects cannot be assigned to certain tasks which they are not willing to implement. In this way, they inherently have a degree of influence over what parts of the software are expanded and improved.

Another dimension of control over technology is agenda setting. Cohen, March, and Olson (1972) argue that participants in an organization are only able to make a restricted number of choices within a certain time frame. Therefore, "attention patterns" within the project matter, implicitly deciding on what choices are discussed and made. In Freenet, a proxy for agenda setting is thread initiation[xx]. By starting a new series of e-mails with a certain topic, a developer or contributor was able define the topic to be discussed within the project. In-depth analysis of single threads revealed that some tended to change the topic during their existence, and the number of threads might not exactly represent the number of topics discussed. However, it is still an acceptable indicator on "what went on" in the project. We found that each developer started 34,9 (standard deviation: 64.65) threads, whereas each of the contributors only began 4.81 (standard deviation: 3.92) threads on average. Developers were more active in setting the agenda within the project, therefore shaping the attention patterns Cohen, March, and Olsen (1972) discussed. Developers were also able to add new and modify existing items on the *ToDo list*, a text file describing potential next steps that are considered a priority by the project. Newcomer contributors often used this list to get a sense of what was important in the project, and therefore it shaped the general direction of the software development.

As the developers decided on the ways in which the software was released during the various stages of development, they could control how the software was used and by whom. Analyzing the threads in emails, two issues we found to be important by their recurring mentions were the release frequency and the release date. A higher release frequency (see Fig. 1 and 2) normally allows the public to test recent source code additions and the developers can thereby get faster feedback on ease of use, functionality, and problems to run the software on various computer platforms (Raymond, 1998). However, each release will be less tested and therefore probably running less stable, as opposed to having fewer but more thoroughly tested releases. Freenet solved this problem by infrequently releasing stable versions

complemented by daily snapshots (a snapshot refers to an automatically packaged software release, derived from the current state of CVS code).

The importance of release scheduling in the Freenet community is illustrated by the following quote by a contributor:

> "*I know that Ian is going to start pushing for a release schedule again now, and I'm beginning to fear that if we don't indulge him soon he will suffer spontaneous human combustion. Certainly, most of the big issues have been weeded out now, and I too am somewhat interested in seeing what happens when people actually use Freenet.*"

The release dates were also important as they determined which features were included in the next version of Freenet as the following e-mail quote shows:

"*What do people think - release now or wait for persistent connections?*" (Ian Clarke)

The third aspect of release management was the packaging and distribution methods being used for the software. Networking software like Freenet, whose proper functioning depends on network externalities, requires a broad user base in order to work reliably (see Section 4). Therefore, making prepackaged software distributions available, which could easily be installed by regular users, was crucial to the success of the project. A comment by a contributor expressed the need for this:

> *"As a Windows user my only beg is that there is a easy way (sic!) for the jabronis to install it that is as simple as gnutella or shoutcast. The more folks banging away at this the better."*

Responding to such calls, some developers spent considerable efforts on building software modules that could provide easy installation on various computer platforms, thereby enhancing functionality as well as the general distribution of Freenet in line with its objectives. In fact, an analysis of the technical characteristics of the software revealed that most of the new developers that joined the project in 2000 made their first commit of software to the CVS around this functionality (von Krogh et al., 2003). By controlling release frequency, release dates, and packaging and distribution mechanisms, developers were able to control to a certain extent who could use the software with what level of stability when running the software on their computers.

In sum, we conclude that developers and, to a certain extent, active list contributors (by their mail messages) through their increasing involvement were rewarded by the communal resource "control over the technology" through the following: direct CVS write access, self appointment of tasks, agenda setting on the discussion list, a *ToDo list*, and release management

Learning opportunities

The third communal resource consists of learning opportunities, defined as collectively accessible opportunities for learning that each individual faces. These opportunities are represented by access to software source code, to experts in a very specialized field, to technical discussions with peers, or to direct feedback to one's own work. The amount and the quality of the learning opportunities increase with the individual involvement in the project.

The literature classifies learning (Bloom, 1956; Kratwohl, Bloom, & Masia, 1964), evaluates learning (Biggs and Collis, 1982), and describes how learning may take place (see Atherton, 2002 for a broad review; Lave and Wenger, 1991; Reynolds, 1965), but the concern often lies with the individual's learning success. Our data does not tell us whether people actually learned in Freenet. Rather, we found conditions and an environment associated with learning at the level of the project without which the project would not have continued (to be able to make a contribution to the emerging software, people first had to understand the emerging software architecture). These conditions include a general flow and exchange of information, feedback and review mechanisms, externalization of knowledge through code and recombination, (Nonaka, 1994) and participation (Lave and Wenger, 1991) in the development process. Etienne Wenger suggested that the emergence of boundaries of a group could be an indicator of the possibilities for collective learning through feedback among its participants (Wenger, 1998: 256):

> *The local depth* these groups *(Note authors' augment) .. provide inevitably creates boundaries, which are … also a sign of learning. But then boundaries themselves become learning opportunities, and the richness of boundary processes becomes a sign of learning as well.*

In other words, active involvement in the group matters, implying that learning opportunities for those who are included in the development process should be higher than for those outside the boundary of the group, or the "peripheral participants". In the context of Open Source these conditions function as a communal resource and we termed them learning opportunities.

There exists a non-linear relationship between cumulated learning opportunities and involvement which starts at very low levels of involvement (lurkers) and peaks at the global maximum with the most highly involved developer in the project (in terms of committed lines of code and number of sent emails). In reality, involvement represents a continuum: It is conceptualized with interest in and commitment towards the project, and it was measured for each actor by the interaction frequency (mails sent to mailing lists) and coding intensity (lines of code written or code modifications to the CVS repository within a certain time). Lurkers are publicly invisible, by definition, and represent the lowest level of involvement. "De-lurking", the first public appearance in the Open Source project, represents a first distinction to the second type of actor, the contributor. The second distinction is CVS write-access that distinguishes the third group: developers. Our categories follow the idea of looking for boundaries of group membership, and classify the actors around an Open Source project in three somewhat arbitrary but practical levels of involvement. For each level of involvement we highlight in this section the applicable learning opportunities.

Learning opportunities cumulate as involvement increases. An additional learning opportunity can either be of a new type or of increased quality, such as more depth of expertise in a conversation or a peer review of the mail message or the software code. The quality of a learning opportunity does not refer to the learning taking place, but to the quality of access to the source of knowledge in the project. A skilled programmer can learn more as a lurker (passively reading through conversations and code) than an incompetent discussion participant can. Nevertheless, the latter has an additional opportunity: interaction. We distinguish two classes of learning opportunities: passive and interaction-based. Our analysis of emails revealed this could be a natural and valid distinction for developers. Consider the following excerpt from an email of a contributor:

> *I wanted to spit out a very quick introduction to everyone. I just joined the freenet-dev list today, and I'm very eager to get involved in the project. I think the concept is absolutely brilliant! I am a software developer with about three years of experience in*

the commercial world writing Java..for my day to day living. My most recent project lasted two years, and involved a distributed architecture based on RMI with cryptography provided by Sun JCE…Hopefully, similar skills are needed somewhere in the FreeNet project. I'll be happy to look through the code and help out where needed, whether it's heads down coding, debugging, writing JavaDoc, or authoring whitepapers. Whatever. Until then, I'll shut up and just absorb the culture a little bit and get my bearings!

Here the contributor indicated his knowledge but also the need to passively learn more about the specific tasks where his expertise in Java programming could be put to use. He decided to step back and lurk until the appropriate task had been found. Our interviews with developers revealed similar approaches. People would lurk and observe the project for a while (up to two months) before they announced their presence on the developer list. However, the learning opportunities intensified as they joined as contributors. As mentioned above, those 26 people we observed joining the project demonstrated considerably higher levels of technical activity than the average contributor, before they were given access as developers to the CVS. They would suggest technical features, bug fixes, and give gifts in the form of software patches. They would receive feedback from other developers and contributors, and change their work and ideas accordingly. Framed in Wenger's (1998) concept of group learning, the boundary of the developer group created its own learning opportunities for those who joined the project at the level of CVS access.

Interaction, in turn, uses two media, human language (usually English) and computer language (code). Hackers hardly ever meet face-to-face and thereby exclude all non-verbal communication except for code (Section 2). One refers to the interaction in conversation and the other to interaction via code. Conversations on mailing lists ranged from superficial to very technical. The dominant medium was written (English) language in mailing lists, and code was rarely copied into an email. The interaction relating to the software code took place on the basis of computer code (and did not necessarily require human language, although developers frequently used written statements alongside the code to help others understand the meaning of it). Both types of interaction can be understood as feedback learning opportunities. Interview data on this "peer-review process", as it is sometimes called, suggests feedback was strong enough to even induce developers and contributors in part to evaluate

their competence (likely intense form of learning). Ian Clarke, the founder of the Freenet project, said:

> *"There's also this intensive continuous peer review process that means that if somebody doesn't have the appropriate skills or understanding, they will very quickly be admonished for it. It's this intensive instantaneous peer review that makes it much easier for people to self-select."*

Lurkers are by definition invisible to the public eye. If measured solely by their interaction their involvement would be zero, but as opposed to anyone not interested in the project, lurkers do get passively involved. They may access all publicly available data including the code base and the discussion lists, current and past. These passive learning opportunities include vast sources of information and codified knowledge. They are never personalized, however, since the lurker does not ask questions.

The range of possible interactions for contributors was large and consequently the learning opportunities differed. A simple one-time comment in the discussion may have resonated very little to not at all. A frequent discussant and skilled programmer without CVS write access (still contributor and not developer in our classification) enjoyed considerably more and better learning opportunities than the one-time contributor. Hence, involvement varied greatly among contributors. All contributors faced the passive learning opportunities of the lurkers, and in addition, the two types of feedback learning opportunities. First, statistically a discussant could expect a reaction on a mailing list posting, as we will see below. Not surprisingly, the reactions differed in quality depending on the nature of the discussion and the nature of the question. Consequently, the quality of the learning opportunity varied from low, associated with superficial discussions, to very high, where the discussion was technical and the discussants displayed extraordinary expertise. Thus, the more involved a contributor got into the depths of technical discussions, the higher was the quality (and expertise) of the reactions on his question or input.

Second, code–based interaction offered another feedback learning opportunity. Many contributors join an Open Source project by fixing bugs or offering software patches as gifts to the project (Bergquist and Ljungberg, 2001; von Krogh et al., 2003). In Freenet, these patches were without exception reviewed by a developer with the authority to add them to the code base of the project (the CVS), and the possible review results offered numerous learning

opportunities for the author. The code was accepted or rejected. If accepted it was sometimes modified by the developer who applied the patch. Both options involved a form of direct feedback for the author's work. The new code may have spurred coding in other modules of the software when adaptation to the new feature was necessary. Remember that Freenet, during the period we researched, grew to 54.000 lines of software code. By learning about how the submitted (accepted) patch triggered development work across the whole software architecture, in other features and modules, the author could gain a better understanding of the overall code. This e-mail interview quote by a developer illustrates how a new patch may trigger discussions among other community members:

> *"Direct feedback usually comes in form of defect reports (closing of existing ones or opening of new ones!) as well as subsequent discussion on mailing lists, particularly if the patch needs more explanation (the resulting explanations then benefit all developers and readers of the mailing lists)."*

Again, the quality of these learning opportunities differed mostly due to the nature of the new code. The larger the impact of the new code on the existing software, the wider was the attention it drew from the contributors and developers. This became clear, for example, in the discussion mentioned above regarding search modules in the software, which developers and some contributors believed compromise anonymity and hence ran counter to the Freenet design goals.

The relatively small group of 30 developers were the most assiduous coders, the most frequent discussants and in general the people who ran the project. Participation in the development list was highly concentrated with four individuals, or 1.1% of the population accounting for 50% of the e-mail traffic. All of these four individuals were developers with the status of committing code to the CVS. The GINI coefficient for message authorship was 0.89 confirming this concentration of activity. In addition to the learning opportunities the contributors have access to, the developers enjoyed the largest share of attention by the other developers and participants and their work was reviewed most frequently. Assuming, like many observers of software development work (e.g. Kohanski, 2000; Pavlicek, 2000) that every line of code written represents a potential software bug and that it needs review and testing, it is obvious that the most important authors of code receive the most reviews of their work from which to learn. On average there were 24 (standard deviation:18) commits per week to the CVS code repository. All 30 developers (8.4% of the total community) added

code to the project. There was a high degree of concentration in the code-writing task with 4 developers (13%) committing 53% of the code to the CVS repository. The GINI coefficient for the code commits was a 0.77 indicating a high degree of concentration in the code commit task.

A similar pattern appeared in the mailing list conversations. Message threads signified that authors of an email brought an important theme of software design to the attention of the project and that this theme sparked further discussion, each new email making a reference to the original e-mail. Thread initiation was similarly concentrated with 10 individuals, 2.8% of the population, accounting for 50% of messages threads initiated. Again, all of these were core-developers in the project. The GINI coefficient for thread initiation by participants was 0.80. A high 78% of the population attempted to initiate dialog at least once on the developer list. Of these attempts only 29 (10.5%) participants did not receive any reply to their initial posting and subsequently did not appear on the developer list again. Choosing a particular topic was a way for developers to lead set the agenda, and efficiently access knowledge of developers and contributors. Developers not only received a lot of individual feedback, they could, due to their central position in the project, even access the majority opinion and get a feel for what the "collective feedback" was (e.g. interview quote by Ian Clarke above regarding release date). Amongst each other, developers sometimes worked in small, and highly specialized teams on a particular problem or module (von Krogh et al., 2003). This intense exchange with other developers offered another high-quality feedback learning opportunity.

A variety of learning opportunities are available to Open Source software development project participants at different levels of involvement. Together they constitute a communal resource which is produced in the software development process, and which is open to those who engage in an Open Source project. There are passive and feedback learning opportunities that in turn vary in quality. An expert replying to a technical question represents a higher-quality learning opportunity than an unqualified comment. The key insight into this communal resource, however, is the positive relationship between involvement and learning opportunities, where only interaction can generate direct feedback from a large number of contributors (in Freenet's case, more than 356 individual contributors and developers , see Section 5). The quality of feedback usually reflects the level of expertise and skill of the programmer regarding the project. And this expertise is closely linked with involvement.

7. Conclusion and implications

Based on resource mobilization theory, we investigated conditions that are sufficient to mobilize programmers to contribute freely to the provision of open source software. An exploratory case study of Freenet, a project developing a sophisticated software for peer-to-peer file sharing over the Internet, showed that the production process of knowledge in an open source software project has as a byproduct communal resources that reward its contributors. In Freenet, the communal resources were reputation, control over technology, and learning opportunities. Whereas these communal resources exist and are perceived as such, we cannot claim them to be exhaustive. We observe them in parallel making it impossible to judge whether each of the communal resources is necessary for the collective action to materialize. However, our study has general implications for resource mobilization theory, research on open source software development and technological innovation.

Resource mobilization theory

There are two distinct contributions to resource mobilization theory. First, in our reformulation of Lerner and Tirole's question (What are the sufficient conditions that mobilize programmers to contribute freely to the provision of a public good?) we employ a resource mobilization framework and find the characteristics of, and the accessibility to three communal resources. Open source software development mobilizes through communal resources the knowledge, time and effort of individual programmers to produce new and innovative software. The organization is emergent and the individuals, different from employees of firms, self-allocate to the tasks they prefer. Rational individuals can be mobilized if they envision certain rewards associated with their costly engagement. In the Freenet project the access to communal resources increased with involvement in the development process. Involvement means dedicating one's own skills and expertise to the project and reflects the effort in software development. Therefore, the communal resources explain the emergence of collective action among rational actors, in spite of active and widespread free riding. Open source software represents a public good since the software is publicly available and as a knowledge good inexhaustible. The communal resources emerge during the production process of this public good as a by-product from the interaction of project contributors and developers, who discuss problems and collectively drive the development process. But the rewards the communal resources provide, are individual. Based on our analysis, we provide an empirical grounding of John Elster's (1986) conjecture on

process-related rewards. Collectively accessible but individually rewarding, communal resources are also a public good for those with the interest and skills to use them. Therefore, they correspond to Mancur Olson's selective incentives. In this combination, the communal resources solve the second order public good problem discussed by Oliver (1993), and therefore open source software development establishes a theoretical typology of collective action (see Oliver, 1993: 293).

This leads us to the second contribution. Resource mobilization theory has been applied to social movements where contributions have neither been distinguished nor selected by the movement. However, because knowledge is both a resource and a goal of the project, open source software represents a theoretical type of collective action where both the value of the public good and the character of its development process require careful selection of available resources. We used the case study of Freenet to explore sufficient conditions for collective action to happen in the setting of open source software development (McPhee, 1995). As mentioned, Freenet did not expend resources on recruiting contributors. Nevertheless, in the year we studied, their number grew to 356 people. Among these individuals, skilled programmers stepped forth and the existing developers observed their knowledge, interest, and commitment to the project, before admitting them as developers. In this sense, open source software projects produce a public good in an inverted form of collective action, through social sampling rather than mass mobilization.

Further research could elaborate a formal model of collective action in open source, and apply it to other situations. For example, academic knowledge creation has a number of parallels to open source software development, where communal resources such as reputation, may be observed in academia as well (see also Stephan, 1996). A broader approach towards communal resources may generate categories of possible communal resources. What other determinants of individual rewards from communal resources exist? How exactly do communal resources emerge and how do they disintegrate or disappear? Can aspects of this typology of collective action be relevant for voluntary action within a firm, whose product is knowledge?

If communal resources explain collective action of the type found here, we may also employ the concept to the analysis of competition in the social movement industries called for by McCarthy and Zald (1977). Comparative case studies of open source projects could identify perceptions of communal resources held by participants across projects. Faster access

to, or higher quality communal resources might induce programmers with scarce knowledge to join specific projects (or social movement). Communal resource may therefore influence the competitive position of social movements. This, in turn, raises questions regarding group composition as a specific problem of resource mobilization. In other words, in a competitive environment, how can a project attract the necessary knowledge to survive?

Research on open source software development and technological innovation

At this stage, in order to advance research on the new phenomenon of open source software development exploratory case studies are needed. Their purpose is foremost to gain familiarity with the new, unfamiliar data and establish measures to be used for large-sample research. Previous studies have established measures of group activity in open source projects (Koch and Schneider, 2000), psychological measures for individual developer motivations (Hertel, Niedner, & Herrmann, 2003), specialization of developers and their level and type of activity (von Krogh et al., 2003), and users' satisfaction with the software product (Franke and von Hippel, 2003). Our research established measures of value to future research on open source software development by operationalizing the constructs of reputation, control over technology, and learning opportunities. Although care was exercised to make the categories non-disjunctive and the constructs operational, measures, items and the external validity of our proposition must be verified across a wider sample of cases.

While a new phenomenon calls for extensive empirical work this paper has also demonstrated that empirical studies need to be guided by theoretical frameworks. Because of our initial research question, we chose resource mobilization theory to guide the inquiry into Freenet. This and other frameworks will serve the purpose of gaining a comprehensive understanding of the causes of collective action, the nature of the open source software project and product. For example, we found that some developers had strong ideological views on the purpose of Freenet. Frameworks other than resource mobilization theory are useful to understand if, and how, passion, grievances, and ideology drive mobilization in open source software development. Theories that explain collective action *not* in terms of interest and cost/benefit analysis, but in terms of changes in social identities associated with radical social or technological change, are more helpful here (Melucci, 1999; Rowley and Moldoveanu, 2003)[xxi]. Possible research questions include: What is the importance of the hackers' orientation towards commercial software development and products for their mobilization? What are the values and ideals underlying conversations among developers? How do values

and norms emerge in an open source software project? What is the role of proselytizing and charismatic leadership in establishing collective action?

Open source software development has produced innovative software on numerous accounts. These software products exist as public goods, whereby the knowledge is available to anyone interested. Hence, open source software development as innovative collective action avoids the social loss problem of innovation seeded and secured by private investments (von Hippel and von Krogh, 2003). From this point of view societies have incentives to foster innovation by collective action as observed in open source alongside the *private innovation model* by entrepreneurs and firms. Future research may show what knowledge and technological innovations can emerge from this typology of collective action. The consequences of a successful private-collective model of innovation could be far-reaching with regards to intellectual property rights, innovation policy and the development of the affected industries. Open source software development may represent an important indicator of a post-industrial society where users develop knowledge and information-based products for their own needs and freely share them with others. Since users' rewards ensue from the production of the public good, in addition to the public good itself, we certainly know it is an interesting and perhaps new form of collective action.

References

Atherton, J. S. 2002. *Learning and Teaching: What is Learning?* http://www.dmu.ac.uk/~jamesa/learning/whatlearn.htm (accessed: 4 Jan 2003).

Austin, R. D. 2001. The effects of time pressure on quality in software development: An agency model. *Information Systems Research*, 12(2): 195-207.

Barley, S. R. 1990. Images of Imaging: Notes on Doing Longitudinal Field Work. *Organization Science*, 1(3): 220-347.

Baron, J. N. & Hannan, M. T. 1994. The Impact of Economics on Contemporary Sociology. *Journal of Economic Literature*, 32(3): 1111-1146.

Benford, R. D. 1993. You Could be the 100Th Monkey - Collective Action Frames and Vocabularies of Motive Within the Nuclear Disarmament Movement. *Sociological Quarterly*, 34(2): 195-216.

Bergquist, M. & Ljungberg, J. 2001. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 11(4): 305-320.

Biggs, J. & Collis, K. 1982. *Evaluating the Quality of Learning: the SOLO taxonomy*. New York: Academic Press.

Bloom, B. S. 1956. *Taxonomy of Educational Objectives, the classification of educational goals – Handbook I: Cognitive Domain*. New York: McKay.

Boehm, A. 2000. Nicht einmal ich koennte Freenet abschalten oder zensurieren. SonntagsZeitung, 139. 10 Dec 2000.

Buechler, S. M. 1995. New Social-Movement Theories. *Sociological Quarterly*, 36(3): 441-464.

Clarke, I. 1999. *A Distributed Decentralised Information Storage and Retrieval System*. Master's Thesis, Division of Informatics, University of Edinburgh.

Clarke,I. 2000. *Interview with E. v. Hippel and K. Lakhani on 10/25*.

Clarke,I., Sandberg,O., Wiley,B., and Hong,T.W. 2000. *Freenet: A Distributed Anonymous Storage and Retrieval System*, Berkeley, CA. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009. 25 Jul 2000.

Cohen, M. D., March, J. G., & Olson, J. P. 1972. A Garbage Can Model of Organizational Choice. *Administrative Science Quarterly*, 17(1): 1-25.

Conner, K. R. & Prahalad, C. K. 1996. A resource-based theory of the firm: Knowledge versus opportunism. *Organization Science*, 7(5): 477-501.

Cusumano, M. A. 1992. Shifting Economies - from Craft Production to Flexible Systems and Software Factories. *Research Policy*, 21(5): 453-480.

Dalle, J. M. & Jullien, N. 2003. 'Libre' software: turning fads into institutions? *Research Policy*, 32(1): 1-11.

Dam, K. W. 1995. Some Economic-Considerations in the Intellectual Property Protection of Software. *Journal of Legal Studies*, 24(2): 321-377.

Demsetz, H. 1967. Towards a theory of property rights. *American Economic Review*, 57(2): 347-359.

Elster, J. 1986. *An Introduction to Karl Marx*. Cambridge: Cambridge University Press.

Eyerman, R. & Jamison, A. 1991. *Social Movements: A cognitive approach*. University Park PA: Penn State Press.

Fehr, E. & Schmidt, K. M. 2000. Fairness, incentives, and contractual choices. *European Economic Review*, 44(4-6): 1057-1068.

Flora, C. B. & Flora, J. L. 1993. Entrepreneurial Social Infrastructure - A Necessary Ingredient. *Annals of the American Academy of Political and Social Science*, 529: 48-58.

Franke, N. & von Hippel, E. 2003. Satisfying Heterogeneous User Needs via Innovation Toolkits: The Case of Apache Security Software. *Research Policy*, 32(7).

freenet.sourceforge.net 2000. *Freenet website*. http://freenet.sourceforge.net.

Friedman, D. & McAdam, D. 1992. Collective identity and activism: Networks, Choices, and the life of a social movement. In A. D. Morris & C. McClurg (Eds.), *Frontiers in Social Movement Theory*: 156-173. New Haven: Yale University Press.

Glaser, B. & Strauss, A. 1967. *The discovery of grounded theory: Strategies of qualitative research*. London: Wiedenfeld and Nicholson.

Gosh, R. A., Glogg, R., Krieger, B., & Robles, G. 2002. *Free/Libre and Open Source software: Survey and study (FLOSS), Part 4: Survey of developers*. http://www.infonomics.nl/FLOSS/report/ (accessed: 20 Nov 2002).

Hartley, J. F. 1995. Case studies in organizational research. In C. Cassell & G. Symon (Eds.), *Qualitative Methods in Organizational Research*: 208-229. London: Sage.

Herman, K. A., Wolfson, M., & Forster, J. L. 1993. The Evolution, Operation and Future of Minnesota Safplan - A Coalition for Family-Planning. *Health Education Research*, 8(3): 331-344.

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).

Himanen, P. 2001. In P. Himanen, L. Torvalds, & M. Castells (Eds.), *The Hacker Ethic and the Spirit of the Information Age*. London: Random House.

Hong,T. 2001. *Performance in decentralized filesharing networks*, Presentation at the O'Reilly Network Conference on Peer-to-Peer computing.

Indyk, D. & Rier, D. A. 1993. Grass-Roots Aids Knowledge - Implications for the Boundaries of Science and Collective Action. *Knowledge-Creation Diffusion Utilization*, 15(1): 3-43.

Jargon File 2002. *The on-line hacker Jargon File (version 4.3.3)*. http://www.catb.org/~esr/jargon/html/index.html.

Knuth, D. E. 1969. *The art of computer programming*. Reading MA: Addison-Wesley.

Koch, S. & Schneider, G. 2000. *Results from Software Engineering Research into Open Source Development Projects Using Public Data*. Wirtschaftsuniversität Wien.

Kohanski, D. 2000. *Moths in the machine*. New York: St. Martin's Press.

Kratwohl, D. R., Bloom, B. S., & Masia, B. B. 1964. *Taxonomy of Educational Objectives, the classification of educational goals– Handbook II: Affective Domain*. New York: McKay.

Lakhani, K. & von Hippel, E. 2003. How Open Source works: "Free" user-to-user assistance. *Research Policy*, Forthcoming.

Lave, J. & Wenger, E. 1991. *Situated Learning: legitimate peripheral participation*. Cambridge: Cambridge University Press.

Lee, S., Moisa, N., & Wiess, M. 2003. *Open source as a signalling device: An economic analysis*. http://opensource.mit.edu/papers/leemoisaweiss.pdf.

Lerner,J. and Tirole,J. 2000. *The Simple Economics of Open Source*. NBER Working Paper Series Working Paper 7600. Cambridge MA, Harvard Business School.

Levy, S. 1984. *Hackers: Heroes of the Computer Revolution*. New York: Anchor Books.

Liebeskind, J. P. 1996. Knowledge, strategy, and the theory of the firm. *Strategic Management Journal*, 17: 93-107.

McAdam, D., McCarthy, J. D., & Zald, M. N. 1988. Social movements. In N. J. Smelser (Ed.), *Handbook of Sociology*: 695-737. Newbury Park CA: Sage.

McCarthy, J. D. & Zald, M. N. 1973. *The Trend of social movements in America*. Morristown NJ: General Learning Press.

McCarthy, J. D. & Zald, M. N. 1977. Resource Mobilization and Social-Movements - Partial Theory. *American Journal of Sociology*, 82(6): 1212-1241.

McPhail, C. & Miller, D. 1973. Assembling Process - Theoretical and Empirical Examination. *American Sociological Review*, 38(6): 721-735.

McPhee, R. D. 1995. Alternate approaches to integrating longitudinal case studies. In G. P. Huber & A. Van den Ven (Eds.), *Longitudinal Field Research Methods*: 186-203. Thousand Oaks: Sage.

Melucci, A. 1999. *Challenging codes: Collective action in the information age*. Cambridge: Cambridge University Press.

Meyer, M. H. & Lopez, L. 1995. Technology Strategy in A Software Products Company. *Journal of Product Innovation Management*, 12(4): 294-306.

Moerke, K. A. 2000. Free speech to a machine? Encryption software source code is not constitutionally protected "speech" under the First Amendment. *Minnesota Law Review*, 84(4): 1007.

Moody, G. 2001. *Rebel code*. Cambridge MA: Perseus Publishing.

Moon, J. Y. & Sproull, L. 2000. Essence of Distributed Work: The Case of the Linux Kernel. *First Monday*, 5(11).

Myers, D. J. 1994. Communication Technology and Social-Movements - Contributions of Computer-Networks to Activism. *Social Science Computer Review*, 12(2): 250-260.

Mykytyn, K., Mykytyn, P. P., Bordoloi, B., McKinney, V., & Bandyopadhyay, K. 2002. The role of software patents in sustaining IT-enabled competitive advantage: a call for research. *Journal of Strategic Information Systems*, 11(1): 59-82.

Nonaka, I. 1994. A Dynamic Theory of Organizational Knowledge Creation. *Organization Science*, 5(1): 14-37.

Nonnecke,B. and Preece,J. 2000. *Lurker demographics: Counting the silent*, The Hague, ACM. Proceedings of CHI 2000.

O'Mahony, S. 2003. Guarding the commons: How community managed software projects protect their work. *Research Policy*, 32(7).

Oberschall, A. 1973. *Social conflict and social movements*. Englewood Cliffs, NJ: Prentice-Hall.

Oliver, P. E. 1980. Rewards and Punishments As Selective Incentives for Collective Action - Theoretical Investigations. *American Journal of Sociology*, 85(6): 1356-1375.

Oliver, P. E. 1993. Formal Models of Collective Action. *Annual Review of Sociology*, 19: 271-300.

Oliver, P. E. & Marwell, G. 1988. The Paradox of Group-Size in Collective Action - A Theory of the Critical Mass .2. *American Sociological Review*, 53(1): 1-8.

Olson, M. 1965. *The Logic of Collective Action: Public Goods and the Theory of Groups*. Cambridge: Harvard University Press.

Oram, A. 2000. *Gnutella and Freenet Represent True Technological Innovation*. http://www.oreillynet.com/pub/a/network/2000/OS/12/magazine/gnutella.html.

Ostrom, E. 1998. A behavioral approach to the rational choice theory of collective action. *American Political Science Review*, 92(1): 1-22.

Pavlicek, R. C. 2000. *Embracing Insanity: Open Source Software Development*. Indianapolis IN: Sams.

Piven, F. F. & Cloward, R. A. 1992. Normalizing collective protest. In A. D. Morris & C. M. Mueller (Eds.), *Frontiers in Social Movement Theory*. New Haven CT: Yale University Press.

Rabin, M. 1993. Incorporating Fairness Into Game-Theory and Economics. *American Economic Review*, 83(5): 1281-1302.

Raymond, E. S. 1998. Homesteading the Noospere. *First Monday*, 3(10).

Raymond, E. S. 1999. *The Cathedral and the Bazaar*. O'Reilly.

Reynolds, B. 1965. *Learning and Teaching in the Practice of Social Work*. New York: Russell and Russell.

Risan, L. 2001. *Hackers produce more than software, they produce hackers (Version 2.1)*. http://folk.uio.no/lrisan/Linux/Identity_games/ (accessed: 12 Dec 2002).

Rowley, T. J. & Moldoveanu, M. 2003. When will stakeholder groups act? An interest- and identity-based model of stakeholder group mobilization. *Academy of Management Review*, 28(2): 204-219.

Schoonhoven, C. B. 2003. Perspectives on open source software development. *Organization Science*, 14(2): 208.

Simon, E. 1996. Innovation and intellectual property protection: The software industry perspective. *Columbia Journal of World Business*, 31(1): 30-37.

Snow, D. A. & Benford, R. D. 1992. Master-frames and cycles of protest. In A. D. Morris & C. McClurg (Eds.), *Frontiers in Social Movement Theory*: 133-154. New Haven: Yale University Press.

Snow, D. A., Zurcher, L. A., & Ekland-Olson, S. 1980. Social Networks and Social-Movements - A Microstructural Approach to Differential Recruitment. *American Sociological Review*, 45(5): 787-801.

Stake, R. E. 1995. Case Studies. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research*: 236-247. Thousand Oaks: Sage Publications.

Stephan, P. E. 1996. The economics of science. *Journal of Economic Literature*, 34(3): 1199-1235.

Strauss, A. & Corbin, J. 1990. *Basics of qualitative research: Grounded theory procedures and techniques*. London: Sage Publications.

Taylor, M. & Singleton, S. 1993. The Communal Resource - Transaction Costs and the Solution of Collective Action Problems. *Politics & Society*, 21(2): 195-214.

Tilly, C. 1978. *From mobilization to revolution*. Reading, MA: Addison-Wesley.

Torvalds, L. & Diamond, D. 2001. *Just for Fun: A story of an Accidental Revolutionary*. New York: Harper Business.

Tuomi, I. 2002. *Networks of Innovation*. Oxford: Oxford University Press.

Useem, B. 1998. Breakdown theories of collective action. *Annual Review of Sociology*, 24: 215-238.

von Hippel, E. & von Krogh, G. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2): 209-223.

von Hippel,E., von Krogh,G., Lakhani,K., and Spaeth,S. 2002. *Innovation in Open Source Software: An intellectual history of Freenet.*

von Krogh, G., Spaeth, S., & Lakhani, K. 2003. Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. *Research Policy*, 32(7).

Waterson, P. E., Clegg, C. W., & Axtell, C. M. 1997. The dynamics of work organization, knowledge and technology during software development. *International Journal of Human-Computer Studies*, 46(1): 81-103.

Wayner, P. 2000. *Free for All: How Linux and the free software movement undercut the high-tech titans*. New York: Harper Business.

Wenger, E. 1998. *Communities of Practice: Learning, Meaning, and Identity*. Camebridge: Camebridge University Press.

Yin, R. K. 1994. *Case Study Research: Design and Methods*. Thousand Oaks: Sage Publications.

Young, G., Smith, K. G., & Grimm, C. M. 1996. "Austrian" and industrial organization perspectives on firm-level competitive activity and performance. *Organization Science*, 7(3): 243-254.

---

[i] When referring to open source projects we also include free or libre software projects which we consider similar enough for our study to be treated as equivalent. For easier readability, however, we will only use the term open source throughout the text.

[ii] Hacker: [originally, someone who makes furniture with an axe] 1. n. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. 2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming. … 8. [deprecated] A malicious meddler who tries to discover sensitive information by poking around. Hence `password hacker', `network hacker'. The correct term for this sense is cracker. (Jargon File, 2002)

[iii] Source code consists of program instructions written in their original form using a programming language readable to humans, such as Pascal, C++, Java or FORTRAN. To execute a program on a computer the source code is translated into machine code using a compiler. Machine code consists entirely of numbers and is only readable by the computer.

[iv] The term copyleft was first used in this context by Richard Stallman who had once received a letter which had several amusing sayings on it, including this one: "Copyleft -- all rights reversed."

[v] Selective incentives derive their name from the way they are put to work: to target individual, potential beneficiaries, encourage their contribution to the public good, and punish their defection.

[vi] In some cases, the production of knowledge can be central to the social movement's success For example, a movement attempting to impose rigid governmental control of research in human genomics through lobbying might fund studies that closely investigate the social and moral implications of such research. For this, they might allocate a part of the monetary resources accumulated to research institutes specializing in this type of studies. Eventually, society might benefit directly from the political actions of the social movement, but indirectly also from knowledge diffusion.

[vii] Michael Cusumano's seminal analysis of commercial software shows that software production evolves within certain parameters, such as product generations, financial resources, time lines, and the capacity of developers. However the process is also fraught with uncertainty, frequent trials, rescheduling, and so on (Cusumano, 1992; Cusumano and Yoffe, 1998; Cusumano and Selby, 1995).

[viii] Research has also uncovered various strategies for recruiting, that fit such a goal (see Benford, 1993; Snow and Benford, 1992)

[ix] As a specific example of a project with an emergent goal, consider the beginnings of what became the Linux Open Source software project. In 1991, Linus Torvalds, a student in Finland, wanted a Unix operating system that could be run on his PC equipped with a 386 processor. Minix was the only software available at that time but it was commercial, closed source, and it traded at USD 150.-. Torvalds found this too expensive, and stared development of a Posix-compatible operating system, later known as Linux. Torvalds did not immediately publicize a very broad and ambitious goal, nor did he attempt to recruit contributors. He simply expressed his private motivation in a message he posted on July 3, 1991, to the USENET newsgroup comp.os.minix (Wayner, 2000: 55) as follows: *Hello netlanders, Due to a project I'm working on (in minix), I'm interested in the posix standard definition.* [Posix is a standard for UNIX designers. A software using POSIX is compatible with other UNIX-based software.] *Could somebody please point me to a (preferably) machine-readable format of the latest posix-rules? Ftp-sites would be nice.* In response, Torvalds got several return messages with Posix rules and people expressing a general interest in the project. By the early 1992, several skilled programmers contributed to Linux and the number of users increased by the day. Today, Linux is the largest Open Source software project extant in terms of number of developers, and in the server software market it is second to Microsoft in terms of servers that use it.

[x] The option of free-riding usually prevents the optimal supply of a public good from the outset by providing negative incentives for the contributors. Free riders as such are of no harm since public goods are by definition non-rivalrous in consumption

[xi] Peer-to-peer software leads to a type of network in which each workstation has equivalent capabilities and responsibilities. In contrast, the traditional network grounds on client/ server software and architecture, in which some computers are dedicated to serve the others. Other peer-to-peer technologies include Gnutella and Napster. Unlike Napster, Freenet does not require a central and operating server for file exchange. Freenet also handles file sharing by storing copies on local servers as the files travel backwards in the network towards the requesting node. This makes the technology more efficient than Gnutella, as when a certain type of information gets requested often, it will be located in the vicinity of the requesting node.

[xii] See at http://geocrawler.org/ or http://sourceforge.net/

[xiii] Lurker: n. One of the 'silent majority' in an electronic forum; one who posts occasionally or not at all but is known to read the group's postings regularly (…).(Jargon File, 2002; see also Nonnecke and Preece, 2000)

[xiv] FAQs can be found at the projects website at: http://freenetproject.org/

[xv] For example by Wired News with Ian Clarke published on October 29, 2002 (http://www.wired.com/news/technology/0,1282,56063,00.html), or by BBC News published on March 12, 2001 (http://news.bbc.co.uk/1/hi/sci/tech/1216486.stm), or by Cnet News on October 28, 2002 (http://news.com.com/2102-1023-963459.html)

[xvi] Also spelled "/. effect" is the phenomenon of a website being virtually unreachable because too many people are hitting it after the site was mentioned in an interesting article on a popular news service like Slashdot.org, linuxtoday.org, or freshmeat.org. (adapted from Jargon File, 2002)

[xvii] A possible source of confusion may stem from the term "resource" as we use it. Whereas "resources" in resource mobilization theory denote the object of mobilization (knowledge, time, effort, labor), the communal resources refer to the potential, communally accessible benefit awaiting the actor who contributes to the collective action, in our case to the Open Source project.

[xviii] patch: 1. n. A temporary addition to a piece of code, usually as a quick-and-dirty remedy to an existing bug or misfeature. A patch may or may not work, and may or may not eventually be incorporated permanently into the program. ... 2. vt. To insert a patch into a piece of code. (Jargon File, 2002)

[xix] One should note, however, that Freenet being a public good would not exclude a potential third party to download the software, and then to start parallel development on her own. This phenomenon where somebody starts up a competing project using existing source code as a basis, is known as *forking*. Forking has been observed in other Open Source projects, such as Open Source Unix (Wayner, 2000), but we have not observed forking in the case of Freenet, and we did not see any mentioning of a threat to fork Freenet.

[xx] A thread consists of a series of e-mails (one mail plus the corresponding answers to it), normally following the same mail heading.

[xxi] We do not distinguish between open source and Fee/libre software projects in our study. But as recent surveys have shown, do especially Free software developer emphasis the differences in their moral and ethical attitude. It might be necessary to examine whether communal resource are equally valid for both types of software development projects (Gosh et al., 2002).