informationresearch

VOL. 13 NO. 1, MARCH 2008

Communal resources in open source software development

Sebastian Spaeth, Stefan Haefliger, Georg von Krogh
Department Management, Technology and Economics, ETH Zurich, Switzerland
Birgit Renzl
Department of Management, University of Innsbruck, Austria

Abstract

Introduction. Virtual communities play an important role in innovation. The paper focuses on the particular form of collective action in virtual communities underlying as Open Source software development projects.
Method. Building on resource mobilization theory and private-collective innovation, we propose a theory of collective action in innovative virtual communities. We identify three communal resources (reputation, control over technology and learning opportunities) that appear as a byproduct while developing open source software.
Analysis. Constructs are derived from exiting literature. Empirical data from Freenet, an open source software project for peer-to-peer software, illustrates both the levels of involvement and the communal resources.
Results & conclusions. Communal resources are able to solve the collective action dilemma for virtual communities. We show that they increase in value for individuals along with their involvement in the community.

Introduction

Virtual communities play an important role in innovation (Moon & Sproull 2000; Lakhani & von Hippel 2003). This paper focuses collective action in virtual communities. The problem of collective action (Hardin 1982) and its possible endogenous solutions in particular, have been suggested by researchers in management (Cabrera & Cabrera 2002), economics (Groves & Ledyard 1977), sociology and law (Heckathorn 1993; Tilly 1978; Taylor & Singleton 1993). Virtual communities, such as open source software projects, have found exceptional ways of

solving the collective action problem. We characterize this solution by complementing the theory of private-collective innovation (von Hippel & von Krogh 2003) with the notion of communal resources in virtual communities and we empirically explore the role of communal resources in collective action. Communal resources provide increasing selective incentives to individuals with deeper involvement in the community.

The private investment model of innovation suggests that 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 through intellectual property law mechanisms such as patents, copyrights and trade secrets. These rights assist innovators in obtaining private returns from their innovation-related investments (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.

Counter to the private investment model, such projects produce software innovations that are public goods characterized by non-excludability and non-rivalry. 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 licences that secure the rights of users to download the code, investigate, modify, apply and redistribute the software. 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 & von Krogh 2003). 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 is 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 contribute to 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 mechanisms to control and safeguard the programming efforts of the beneficiary. However, free riding is also inauspicious to the provision of the public good: 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 & Zald 1977). In this paper, against the backdrop of resource mobilization theory, we pose the following research question: What are the sufficient conditions that mobilize programmers to contribute freely to the provision of a public good? Two steps are necessary to answer this question: first, combining concepts from resource mobilization and a theory of private-collective innovation outlines the particular form of collective action at work in virtual communities, particularly the importance of communal resources. Second, a case study explores the properties of communal resources in open source software development. A closer examination of the 'fertile middle ground where incentives for private investment and collective action can coexist' (von Hippel & von Krogh 2003: 213) combined with insights from psychological studies of developers' motivations (Hertel *et al.* 2003) reveals that open source

software development contains process-related rewards (Elster 1986). These rewards, or communal resources, represent a public good of the second order (Oliver 1980) and resolve the collective action dilemma referred to earlier. Communal resources are collectively produced by the project and provide individual rewards for developers and, as we show empirically, these rewards increase with the developers' involvement in the open source software project. The paper is organized as follows: the next section briefly discusses resource mobilization theory and the literature that leads to an understanding of collective action in virtual communities. The section *Research methods* provides an overview of our research methods and introduces the case of Freenet . The section *Communal resources* identifies communal resources and illustrates their properties in the case of Freenet. The *Discussion* concludes the paper by discussing limitations and further implications for organizational theory and research.

Theory

The theory section outlines the particular form of collective action that takes place in innovative virtual communities. Building on resource mobilization theory, we examine a form of private-collective innovation and focus on the organizational conditions that mobilize contributors to open source software projects. 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 & Sproull 2000) and the Apache project (Lakhani & 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.

Developed in opposition to a Durkheimian 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 & Cloward 1992; Useem 1998). The resource mobilization perspective argues that organization underlines successful collective action projects (Tilly 1978), including the professionalisation of such organizations, the career patterns of social movement personnel and the emerging social movement industries (McAdam *et al.* 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 & 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 & 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 & McAdam 1992; Oliver 1980)[i]. 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 analysed 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 labour (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 rises, 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 their activities and thereby generate considerable value for society (e.g., Eyerman & Jamison 1991; Flora & Flora 1993; Herman *et al.* 1993; Indyk & Rier 1993; Myers 1994)[ii]. However, resource mobilization theory, thus far, has not considered collective action that has the production of knowledge as the primary organization goal, which is the case for a Free software project (software code) (see von Hippel & von Krogh 2003). For example, a project that develops software for 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 labour 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 *et al.* 2002) and they spend part of either 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[iii]. 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 & 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 behaviour (McCarthy & Zald 1977; 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 & Miller 1973; Snow & Benford 1992; Snow *et al.* 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 & Marwell 1988; Taylor & Singleton 1993)[iv].

Because open source projects need knowledgeable developers one would expect that they would engage in precise goal 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. 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 example, a Freshmeat.net link points to Gnofract 4D site, where the 'Help wanted' link leads to:

> Please consider contributing to Gnofract 4D. If you aren't a programmer, you can still contribute by reporting bugs, writing formulas or coloring methods, creating gradients, or just by posting your images to the User Gallery to inspire others.

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 Freshmeat.net). However, potential participants can access information about the project by searching the Internet. In open source software projects most communication is through the Internet, people rarely meet face-to-face. Some people participate under pseudonyms concealing their real identity, thereby obscuring their interests to the entrepreneur. Even under these seemingly adverse conditions, projects such as Linux or Apache have shown that they can be successful in attracting large groups, in some cases thousands, of contributors.

Finally, it is interesting to note that these 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 & 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:

> ... 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. (1986: 132)

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[v]. This implies that the development process may be just as important as the (envisioned) final product. For example, GNU/Linux has evolved since 1991 and its contributors seem to be rewarded quickly for their contributions.

The notion of communal resources points to the value created by social relations and norms (Taylor & Singleton 1993; von Krogh 2002) when groups are facing transaction costs that could prevent their effective coordination. Communal resources take the role of a public good of second order (i.e., represent the selective incentives necessary to enable collective action) in that they provide private benefits to individuals who contribute to the collectively accessible public good. Researchers on virtual communities know well that enabling conditions include both communication infrastructure (Moon & Sproull 2000) and social factors such as trust (Jarvenpaa & Leidner 1998)

In line with Elster's conjecture, we propose the following:

> **P1: The production process of knowledge in an open source software project has, as a by-product, 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 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 and accessibility of the communal resources are the sufficient, organizational conditions that mobilize contributors to the public good produced in virtual communities. Based on our proposition, we empirically identify and explore the properties of the communal resources in the case of Freenet, but first we turn to the research method and introduce the case.

Research methods

In this section we present the method used to study communal resources in the open source software setting. We decided to use a case study approach (Yin 1994: 30; Hartley 1995; McPhee 1995), which is, first, especially appropriate in new topic areas and, secondly, allows us to understand the dynamics present within this setting (Eisenhardt 1989: 534). The purpose of our case study is to investigate communal resources built during the development process and rewarding contributors to an open source software project. We have decided to study the case of Freenet, an extraordinary open source software project in regard to initiation of the project; how it works, a radical innovation in peer-to-peer software; and considering the newness of the project as will be outlined in more detail below. The case study explores our proposition regarding the sufficient conditions for routine collective action to happen in the novel setting of open source software development. In this research we combined various methods of data collection such as interviews, archives, observations of mailings lists and committed source codes. Qualitative and quantitative data both complement the insights into the functioning of the Freenet project. In the following we first introduce Freenet and then we outline the research design in more detail.

The case of Freenet

Freenet is software that allows the publishing and obtaining of information through the Internet without the possibility of censorship, i.e., the network is completely decentralized and publishers as well as consumers remain anonymous. It is a peer-to-peer network which can be used for publishing Websites, communicating through message boards, as well as for content distribution. It resembles an anonymous Internet within the Internet. Ian Clarke started the Freenet project as a fourth year computer science student. He defined the overall goals of Freenet as follows (Clarke 1999):

- 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[vi]. Information moves from parts of the Internet where it is in low demand to areas where demand is greater.

Freenet was first released in April 2000. Since then, the Freenet software has been downloaded more than 1,400,000 times. This shows the significant public interest in this software, which also possibly triggered the interest of many potential new developers to the project. When the first public release of Freenet (0.1 Beta) was made, this resulted in a steep increase in source code contributions and sparked the interest and entry of new developers into the project. Collective action in Freenet includes the discussions on the development mailing list that accompanies the production of the software code. On average the Freenet project consisted of 45 (standard deviation = 21) active participants a week. This number was achieved around the first public release date of the project and after that it remained fairly stable. In 2000, 356 individuals participated in the main Freenet developer discussion list, generating 11,210 e-mail messages in 1,714 message threads.

The case of Freenet represents an extraordinary open source software project which helps us to understand our process of interest, i.e., identifying communal resources that are able to solve the collective action dilemma in virtual communities. Typically extreme situations and extraordinary types of cases in which the process of interest is 'transparently observable' (Eisenhardt 1989: 537) are chosen. Freenet has been selected for three reasons in our research. First, the development of Freenet hinges on knowledge of cryptography, which is considered a rare resource among experts on software development. It pursues the ambitious goal of creating new knowledge rather than improving already existing software. In contrast to Linux, Freenet was launched not on the basis of workable code written by an entrepreneur (Lerner & Tirole 2000), but rather theory driven by the master thesis of its founder Ian Clarke, outlining the theoretical principles of anonymous peer-to-peer computing (Clarke 1999). Second, Freenet symbolizes a radical innovation of peer-to-peer software[vii] (Boehm 2000; Oram 2000), which bears increased cost of contributing. Freenet has no template of software architecture available, such as Unix for Linux (Wayner 2000). Hence, it can be reasoned that those involved in the project would not have an easy understanding and access to the technology. 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 *et al.* 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 & Zald 1977).

Research design

Data were collected for the year 2000, the first year of Freenet development. The first year is particularly important for collective action as communal resources are built up and contributors need to be attracted. This year involved the mobilization of twenty-six developers to a total of thirty. Given a fairly stable group of developers, it can be reasoned that the following years' data would not provide much additional insight regarding mobilization of developers. Freenet in its first year of existence symbolizes a critical pilot case with respect to studying our proposition (Glaser & Strauss 1967; Stake 1995; Strauss & Corbin 1990; Yin 1994). Combining multiple data collection methods we used expert interviews, mailing lists, source code and archives in our case study research, which will be outlined in the following.

We started our case study with in-depth interviews and a questionnaire. First, we identified thirteen Freenet core developers on the project's homepage and conducted telephone interviews. The interviews of one to two hours each were carried out in the period of October, 2000 to January, 2001 and March to May, 2001 in three rounds. They followed semi-structured interview guidelines including topics such as developer background information, overall structure of the project, reason for joining and working on the project, rewards, specialization and particular challenges in the project. All interviews were recorded and transcribed to facilitate data analysis. To obtain further insights on communal resources we conducted an additional survey sending a questionnaire electronically to participants in the project in May 2003.

Second, we analysed the contributors' public e-mail conversations gathered in Freenet's mailing lists. Focusing on contributions to the technical development of Freenet, we collected e-mail data from the 'development' list where contributors discuss themes regarding the next release of Freenet on the Internet, its design, upcoming architecture and other technical aspects. We gathered those messages in a database including contributor identity, date and time of posting a message, mails responding to the message and mail content for the period January 1 - December 26 2000, on a month-by-month basis. The database consisted of approximately 12,000 single e-mail messages from 356 unique participants. However, no attempt was made to extend the analysis to measure the number of lurkers on this list[viii].

In the third step, we analysed the committed source code within the Concurrent Versions System (CVS). CVS is a public version control tool that synchronizes work and keeps track of changes in the source code performed by developers working on the same file. The CVS archives its version-control information in a directory hierarchy on a central server, called the Repository, which is separate from the user's working directory. The repository allows files to be added or removed easily, or for information about a set of files to be sought. The CVS also archives the developers' comments documenting their work. While source code and comments are publicly available, only a limited number of about thirty core developers are able to commit source code to the CVS in Freenet. We found that source code commits represented a crucial data pool, since project progress was tracked by the progress of source code modifications. Data collection in the period January 1 - December 26 2000, consisted of 1,244 source code commits from the thirty developers. Excluding the initial revisions of code a total of 54,000 lines of software code was added.

Fourth, we completed our case analyses by taking into account publicly available documents related to Open Source in general and to the project in particular. We studied the Freenet

project Web pages (e.g., the Frequently Asked Questions), Ian Clarke's master thesis (1999), journals on peer-to-peer software and interviews with the core developers[ix], a working paper describing the Freenet technology (Clarke *et al.* 2000) and a presentation on results of a simulation of the software (Hong 2001). We contacted the Freenet developers by e-mail in case of any queries, ambiguity, or lack of data. In the next section we will outline how we proceeded with data analysis regarding rewards for those who contribute this substantial amount of development work to Freenet.

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 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). Here, 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 (1990) called *systematization* of topics emerging from his observations of radiology departments. 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[x]. In subsequent iterations of analysis we sought to make the three constructs operational.

Involvement

For the analysis of Freenet we distinguished the participant's level of involvement into three distinct categories: lurkers, contributors and developers.

Lurkers eavesdrop on mailing or discussion lists without posting (Nonnecke & 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 *et al.* 2003). As mentioned in the next section, 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 through 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 make exorbitant demands for features (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[xi].

Members of the third group, developers, are defined by the permission to modify the Freenet source code in the code repository. 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. 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 case of Freenet, thirty developers were granted developer status during the data collection period.

In a project interview, Freenet founder Ian Clarke stated that involvement is generally associated with increased technical skills, merit and a sense of responsibility. It should be noted that within the contributor and developer category, the level of involvement varies significantly. Within these groups involvement can be measured through, for example, the number of sent e-mails and the number of code contributions. Among the thirty developers we found some who were particularly active, e.g., the top four developers produced 50% of all e-mail traffic.

In the following, we 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.

Level of Involvement	**Communal Resource**		
	Reputation	**Control over technology**	**Learning Opportunities**
Lurker	No	No	Passive
Contributor	Low	Limited	Passive & Feedback
Developer	High	Extensive	Passive & Quality Feedback

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' judgments 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 projects consist of experts in computer programming and the daily work on coding and the technical discussions for a particular project hardly ever resonate 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 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 labour market argument, Lerner and Tirole (2000) and Lee *et al.* (2003) suggest that developers who build a strong reputation in the Open Source projects would enhance their human capital and so would raise their value to a potential employer. Although intriguing, the interviews we conducted with the developers of Freenet did not confirm that they contributed in order to raise their value in the labour market for software engineers. For example, a developer, who calls himself 'Mr. Bad', used fictitious identities to conceal his real identity and developers often obscure their affiliations with software firms. Rather, it seems the direct effect of reputation among peer developers was important to Freenet developers, which is consistent with assertions in the literature (Himanen 2001; Lerner & 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 core 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 Torvalds. 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), the foremost example being Linus Torvalds (Torvalds & 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, which 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 were sent to core-developers by various contributors (before they received 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 most of the time was, 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), each of whom wrote, on average, three times more mails than the next frequent discussant. The nineteen 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 one in twenty lauds was addressed to a regular contributor, whereas the other nineteen 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 core developers, who were mentioned regularly. Most 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 (core developers and contributors) 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.

> **P2: The individual benefits from reputation increase with involvement in the community.**

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 is able to use the software through various means. As mentioned only thirty 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 functionality of the Freenet technology[xii].

We found, for example, that some developers had strong ideological interests associated with Freenet, particularly 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 make the location of information easier for Freenet users could at the same time compromise anonymity of the sender and receiver of information and was deemed against the Freenet design goals by some 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, 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. Imagine a malevolent developer intentionally committing code to the CVS, which caused the software to malfunction. In this 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 quotation 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 quotation also shows that developers controlled what was being worked on in the technology through the self-appointment of tasks. This seems typical for open source projects (Schoonhoven 2003) as, unlike in a software firm, contributors and developers in such projects cannot be assigned to certain tasks they are not willing to implement. In this way, they have an inherent degree of influence over which parts of the software are expanded and improved.

Another dimension of control over technology is concerned with agenda setting. Cohen *et al.*

(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, agenda setting can be measured through the percentage of initiated threads on the development mailing list[xiii]. By starting a new series of e-mails on a certain topic, a developer or contributor was able to 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 of '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 *et al.* (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 how the software was released during the various stages of development, they could control how the software was used and by whom. Analysing the threads in e-mails, two issues we found to be important by their recurring mentions were the release frequency and the release date. A high release frequency normally allows the public to test recent source code additions and the developers, thereby, can get faster feedback on new functionality or problems with the new version (Raymond 1999). However, each release will be less tested and probably, therefore, 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 quotation from 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 quotation 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. Therefore, making pre-packaged 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 in 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 who joined the project in 2000 made their first commit of software to the CVS on 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 e-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.

> **P3: The individual benefits from control over the technology increase with involvement in the community.**

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*et al.* 1964), evaluates learning (Biggs & Collis 1982) and describes how learning may take place (see Atherton 2005 for a broad review; Lave & Wenger 1991; Reynolds, 1965), but the concern often lies with the individual's learning success. Our data do 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 & Wenger 1991) in the development process. 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:

> The local depth these groups... 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. (Wenger 1998: 256)

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 e-mails). In reality, involvement represents a continuum: it is conceptualized as 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 involved in 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 e-mails revealed this could be a natural and valid distinction for developers. Consider the following excerpt from an e-mail 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 twenty-six 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. One refers to the interaction in conversation and the other to interaction through 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 e-mail. 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 & Ljungberg 2001; von Krogh *et al.* 2003). In Freenet, these patches, without exception, were 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 quotation 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 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 thirty 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 twenty-four (standard deviation 18) commits a week to the CVS code repository. All thirty 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 four 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 e-mail brought an important theme of software design to the attention of the project and that this theme sparked further discussion, each new e-mail making a reference to the original e-mail. Thread initiation was similarly concentrated with ten 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 dialogue at least once on the developer list. Of these attempts only twenty-nine (10.5%) participants did not receive any reply to their initial posting and did not appear on the developer list again. Choosing a particular topic was a way for developers to set the agenda and efficiently access knowledge of developers and contributors. Developers not only received a lot of individual feedback, they could, because of their central position in the project, even access the majority opinion and get a feel for what the 'collective feedback' was (e.g., interview quotation by Ian Clarke above regarding release date). Amongst themselves, 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 is 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). 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.

> **P4: The individual benefits from learning opportunities increase with involvement in the community.**

Discussion and conclusions

While a new phenomenon calls for extensive empirical work, this paper also demonstrates that empirical studies need to be guided by theoretical frameworks. We chose resource mobilization theory to guide the inquiry into Freenet, investigating conditions that are sufficient to mobilize programmers to contribute freely to the provision of a public good. Our exploratory case study showed that the knowledge production process in an open source software project itself created several communal resources as a byproduct. These reward contributing individuals for their efforts. In Freenet, the communal resources were reputation, control over technology and learning opportunities. Our study has general implications for resource mobilization theory, collective action, research on open source software development and technological innovation.

There are two main 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, identifying the characteristics of and the accessibility to three communal resources. Through these, open source software development mobilizes the knowledge, time and efforts of individual programmers to produce new and innovative software. Communal resources emerge during the production process of this public good as a by-product through the collective interaction of project contributors and developers during the development process. But the rewards that they provide are only accessible for the individual contributors. This solves the theoretical puzzle of mobilizing 'rational' individuals without the need to assume a sort of intrinsic or altruistic motivation, or to reduce motivation to a labour market value argument.

The concept of communal resources explains the emergence of collective action among rational actors contributing to a public good in spite of active and widespread free riding. Based on our analysis, we thus provide some empirical grounding for 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 take advantage of them. Therefore, communal resources provide and correspond to Olson's selective incentives. This solves the second order public good problem discussed by Oliver (1993). Therefore, open source software development establishes a theoretical typology of collective action (see Oliver 1993: 293).

This leads to the second contribution to resource mobilization theory: it has been applied to social movements where contributions have neither been distinguished nor selected by the movement. However, 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, because knowledge is both a resource and a goal of the project. 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 existing developers observed their interest, commitment and contributions 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 can explain collective action in an open source software setting, 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?

Previous studies in open source software have established measures of group activity in open source projects (Koch & Schneider 2000), psychological measures for individual developer motivations (Hertel *et al.* 2003), specialization of developers and their level and type of activity (von Krogh *et al.* 2003) and users' satisfaction with the software product (Franke & von Hippel 2003). Our research establishes and identifies the constructs of reputation, control over technology and learning opportunities as instantiations of communal resources. In the Freenet project we found that access to communal resources did increase with a higher level of involvement. Further research can use these measures to validate and extend the existence of communal resources across a wider sample of open source software and other knowledge based projects.

Open source software products exist as public goods, whereby the knowledge is available to anyone interested, avoiding the social loss problem of innovation seeded and secured by private investments in the private innovation model (von Hippel & von Krogh 2003). Therefore, societies have incentives to foster innovation through collective action as observed in open source software, as a form of the private-collective innovation model. If communal resources provide the necessary selective incentives to mobilize individuals, the consequences of such a successful private-collective model of innovation could be far-reaching with regards to intellectual property rights, innovation policy and the development of knowledge-based 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.

Acknowledgements

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

Footnotes

i. 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.

ii. 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.

iii. 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).

iv. Research has also uncovered various strategies for recruiting, that fit such a goal (see Benford 1993; Snow & Benford 1992)

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

vi. Also spelled '/. effect' , this is the phenomenon of a Website being virtually unreachable because too many people are hitting it after the site has been mentioned in an interesting article on a popular news service like Slashdot.org, linuxtoday.org, or freshmeat.net (adapted from Jargon File 2002).

vii. 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.

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

ix. For example by Wired News with Ian Clarke published on October 29, 2002, or by BBC News published on March 12, 2001, or by Cnet News on October 28, 2002

x. 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, labour), 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.

xi. 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. To insert a patch into a piece of code. (Jargon File 2002)

xii. One should note, however, that Freenet being a public good would not exclude a potential third party downloading the software and then starting 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 mention of a threat to fork Freenet.

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

References

- Atherton, J.S. (2005). *Learning and teaching: angles on learning, particularly after the schooling years.* Retrieved 30 October, 2007 from: http://www.learningandteaching.info/learning/.
- Barley, S. R. (1990). Images of imaging: notes on doing longitudinal field work. *Organization Science*, **1**(3), 220-347.
- Baron, J. & Hannan, M. (1994). The impact of economics on contemporary sociology. *Journal of Economic Literature*, **32**(3), 111-1146.
- 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, NY: Academic Press.
- Bloom, B.S. (1956) Taxonomy of educational objectives, the classification of educational goals. Handbook I: cognitive domain. New York, NY: McKay.
- Boehm, A. (2000, December, 10). Nicht einmal ich koennte freenet abschalten oder zensurieren. [Not even I could switch off or censor freenet.] *SonntagsZeitung*.
- Buechler, S.M. (1995). New social-movement theories. *Sociological Quarterly*, **36**(3), 441-464.
- Cabrera, Á. & Cabrera, E.F. (2002). Knowledge-sharing dilemmas. *Organization Studies*, **23**(5), 687-710.
- Clarke, I. (1999). *A distributed decentralised information storage and retrieval system*. Unpublished master's thesis, University of Edinburgh, Edinburgh, Scotland.
- Clarke, I., Sandberg, O., Wiley, B. & Hong, T.W. (2000). Freenet: a distributed anonymous storage and retrieval system. *In Hennes Fedderath, (Ed.). Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000. Proceedings* (pp. 46-66). Berlin: Springer. (Lecture Notes in Computer Science, Volume 2009/2001)
- Cohen, M.D., March, J.G. & Olson, J.P. (1972). A garbage can model of organizational choice. *Administrative Science Quarterly*, **17**(1), 1-25.
- Coleman, J.S. (1973). *The mathematics of collective action*. Chicago, IL: Aldine.
- 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.
- Demsetz, H. (1967). Towards a theory of property rights. *The American Economic Review*, **57**(2), 347-359.
- Eisenhardt, K.M. (1989). Building theories from case study research. *Academy of Management Review*, **14**(4), 532-550.

- 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**(1), 48-58.
- Franke, N. and von Hippel, E. (2003). Satisfying heterogeneous user needs via innovation toolkits: the case of apache security software. *Research Policy*, **32**(7), 1199-1215.
- Friedman, D. & McAdam, D. (1992). Collective identity and activism: networks, choices and the life of a social movement. In: A. Morris, A. and C. McClurg Mueller, (Eds.) *Frontiers in social movement theory*. (pp. 156-173). New Haven, CT: 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. Deliverable D18: FINAL REPORT. Part IV: Survey of Developers.* Retrieved 20 November 2002 from http://www.infonomics.nl/FLOSS/report/Final4.htm
- Groves, T. & Ledyard, J.O. (1977). Optimal allocation of public goods: a solution to the 'free rider' problem. *Econometrica*, **45**(4), 783-809.
- Hardin, R. (1982). *Collective action*. Baltimore, MD: John Hopkins University Press.
- Harhoff, D., Henkel, J. & von Hippel, E. (2003). Profiting from voluntary information spillovers: how users benefit by freely revealing their innovations. *Research Policy*, **32**(10), 1753-1769.
- Hartley, J.F. (1995). Case studies in organizational research. In: C. Cassell & G. Symon (Eds.), *Qualitative methods in organizational research*, (pp. 208-229). London: Sage.
- Heckathorn, D.D. (1993). Collective action and group heterogeneity: voluntary provision versus selective incentives. *American Sociological Review*, **58**(3), 329-350.
- 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. & Hermann, S. (2003). Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, **32**(7), 1159-1177.
- Himanen, P. (2001). *The hacker ethic and the spirit of the information age*. London: Secker &Warburg.
- von Hippel, E. & von Krogh, G. (2003). Open source software and the 'private-collective' innovation model. *Organization Science*, **14**(2), 209-223.
- Hong,T. (2001). *Performance in decentralized filesharing networks*, presentation at the O'Reilly Peer-to-Peer Conference Westin St. Francis Hotel — San Francisco, California, February 14-16, 2001. Retrieved 29 October, 2007 from ftp://ftp.ora.com/pub/conference/p2p2001/1197/hong_1197.ppt
- 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.

- The Jargon file (version 4.4.7). (n.d.). Retrieved 29 October, 2007 from http://www.catb.org/~esr/jargon/html/index.html.
- Jarvenpaa, S.L. & Leidner, D.E. (1998). Communication and trust in global virtual teams. *Journal of Computer-mediated Communication*, **3**(4), 1-28.
- 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. Vienna: Wirtschaftsuniversität Wien.
- Kohanski, D. (2000) *Moths in the machine*. New York, NY: 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, NY: McKay.
- von Krogh, G. (1998). Care in knowledge creation. *California Management Review*, **40**(3), 133-153.
- von Krogh, G. (2002). The communal resource and information systems. *Journal of Strategic Information Systems*, **11**(2), 85-107.
- von Krogh, G., Spaeth, S. & Lakhani, K. (2003). Community, joining and specialization in open source software innovation: a case study. *Research Policy*, **32**(7), 1217-1241.
- Lakhani, K.R. & von Hippel, E. (2003) How open source works: "free user-to-user assistance". *Research Policy*, **32**(7), 923-943.
- 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*. Frankfurt: Johann Wolfgang Goethe-Universität. (Finance & Accounting Working Paper No. 102) Retrieved 29 October, 2007 from http://opensource.mit.edu/papers/leemoisaweiss.pdf.
- Lerner, J. & Tirole, J. (2000). *The simple economics of open source*. Cambridge, MA: National Bureau of Economic Research. (NBER Working Paper No. 7600) Retrieved 29 October, 2003 from http://tinyurl.com/3ccgwu
- Levy, S. (1984). *Hackers: heroes of the computer revolution*. New York, NY: Anchor Books.
- Liebeskind, J.P. (1996). Knowledge, strategy and the theory of the firm. *Strategic Management Journal*, **17**(Winter special issue), 93-107.
- McAdam, D., McCarthy, J. & Zald, M.N. (1988) Social movements. In N. Smelser, (Ed.). *Handbook of sociology*, (pp. 695-737). Newbury Park, CA: Sage.
- McCaffrey, D., Faerman, S. & Hart, D.W. (1995) The appeal and difficulties of participative systems. *Organization Science*, **6**(6), 603-627.
- McCarthy, J. & Zald, M.N. (1973). The trend of social movements in America, Morristown, NJ: General Learning Press.
- McCarthy, J. & Zald, M.N. (1977). Resource mobilization and social-movements - partial theory. *American Journal of Sociology*, **82**(6), 1212-1241.
- McPhail, C. and 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, (pp. 186-203). Thousand Oaks, CA: Sage.
- Melucci, A. (1999). *Challenging codes: collective action in the information age*. Cambridge: Cambridge University Press.
- Merton, R.K. (1973) *The sociology of science: theoretical and empirical*

investigations. Chicago, IL: University of Chicago 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.
- Monge, P.R., Fulk, J., Kalman, M.E., Flanagin, A.J., Parnassa, C. & Rumsey, S. (1998). Production of collective action in alliance-based interorganizational communication and information systems. *Organization Science*, **9**(3), 411-433.
- 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). Retrieved 29 October, 2007 from http://www.firstmonday.org/issues/issue5_11/moon/
- 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. & Preece, J. (2000) Lurker demographics: counting the silent. In *Proceedings of the SIGCHI conference on Human factors in computing systems, The Hague, The Netherlands*, (pp. 73-80). New York, NY: ACM Press.
- O'Mahony, S. (2003). Guarding the commons: how community managed software projects protect their work. *Research Policy*, **32**(7), 1179-1198
- 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. *American Sociological Review*, **53**(1), 1-8.
- Olson, M. (1965). *The logic of collective action*. Cambridge, MA: Harvard University Press.
- Oram, A. (2000). *Gnutella and freenet represent true technological innovation*. Sebastopol, CA: O'Reilly Network. Retrieved 15 October, 2007 from http://tinyurl.com/2jqtl8.
- Ostrom, E. (1998). A behavioural 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: Sams..
- Piven, F.F. & Cloward, R.A. (1992). Normalizing collective protest. In A. Morris & C. McClurg Mueller, (Eds.). *Frontiers in social movement theory*, (pp. 301-325). 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. (1999). *The cathedral and the bazaar: musings on linux and open source by an accidental revolutionary*. Sebastopol, CA: O'Reilly.
- Reynolds, B. (1965). *Learning and teaching in the practice of social work*. New York,

NY: Russell and Russell.
- Risan, L. (2001). *Hackers produce more than software, they produce hackers*. (Version 2.1). Retrieved 12 December 2002 from http://folk.uio.no/lrisan/Linux/Identity_games/.
- 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.
- 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*, (pp. 133-154) New Haven, CT: 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*, (pp. 236-247). Thousand Oaks, CA: Sage Publications.
- Strauss, A. & Corbin, J. (1990). Basics of qualitative research: grounded theory procedures and techniques. London: Sage Publications.
- Stephan, P. (1996). The economics of science. *Journal of Economic Literature*, **34**(3), 1199-1235.
- Taylor, M. & Singleton, S. (1993). The communal resource: transaction costs and the solution of collective action problems. *Politics & Society*, **21**(2), 195-214.
- Tilly, Charles (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, NY: 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.
- 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*. New York, NY: Harper Business..
- Wenger, E. (1998). *Communities of practice: learning, meaning and identity*. Cambridge: Cambridge University Press.
- Yin, R.K. (1994). *Case study research: design and methods*. London: Sage.
- 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.

How to cite this paper

Spaeth, S., Haefliger, S., von Krogh, G. & Renzl, B. (2008). "Communal resources in open source software development" *Information Research*, **13**(1) paper 332. [Available 4 November, 2007 at http://InformationR.net/ir/13-1/paper332.html]

Find other papers on this subject