ELSEVIER

Research Policy 32 (2003) 1217–1241

research policy

www.elsevier.com/locate/econbase

Community, joining, and specialization in open source software innovation: a case study

Georg von Krogh [a], Sebastian Spaeth [a,*], Karim R. Lakhani [b]

[a] *Institute of Management, University of St. Gallen, Dufourstrasse 48, CH-9010 St. Gallen, Switzerland*

[b] *MIT Sloan School of Management, 50 Memorial Drive, Cambridge, MA 02139, USA*

Abstract

This paper develops an inductive theory of the open source software (OSS) innovation process by focussing on the creation of Freenet, a project aimed at developing a decentralized and anonymous peer-to-peer electronic file sharing network. We are particularly interested in the strategies and processes by which new people join the existing community of software developers, and how they initially contribute code. Analyzing data from multiple sources on the Freenet software development process, we generate the constructs of "joining script", "specialization", "contribution barriers", and "feature gifts", and propose relationships among these. Implications for theory and research are discussed.

Keywords: Open source software; Innovation; Community; Collective action; Virtual teams

1. Introduction

The production of open source software[1] (OSS) results in the creation of a public good that is non-rival, i.e. users' utility from the software are independent, and non-exclusive, i.e. no individual or institution can be feasibly withheld from its usage (Lerner and Tirole, 2002). OSS development represents a "private-collective" model of innovation where developers obtain private rewards from writing code for their own use, sharing their code, and collectively contributing to the development and improvement of software (von Hippel and von Krogh, 2003). This model explains the existence of the open source phenomenon, but leaves open a number of questions about the innovation process that requires in-depth empirical research. An assertion in the private-collective model, and much of the writing on OSS is that the success of a project in terms of producing the software relates to the growth in the size of the *developer community*; people who contribute to the public good of open source software by writing software code for the project (Moody, 2001; Raymond, 1999; Sawhney and Prandelli, 2000; Wayner, 2000). However, joining a developer community may not be costless. Software development is a knowledge-intensive activity that often requires very high levels of domain knowledge, experience, and intensive learning by those contributing to it (Pliskin et al., 1991; Waterson et al., 1997). Fichman and Kemerer (1997) found that in commer-

* Corresponding author. Tel.: +41-71-224-23-63; fax: +41-71-224-23-55.
E-mail addresses: georg.vonkrogh@unisg.ch (G. von Krogh), sebastian.spaeth@unisg.ch (S. Spaeth), karim.lakhani@sloan.mit.edu (K.R. Lakhani).

[1] Software that comes with source code and a usage license that allows for modification and further redistribution of the source code by any user.

0048-7333/03/$ – see front matter
doi:10.1016/S0048-7333(03)00050-7

cial software development, complex technologies can erect significant barriers of understanding and contribution, to both users and developers of the software, and the integration of newcomers can be arduous. As the technology grows more complex, only a few people who have been actively involved in its development over a certain period of time might fully understand the software architecture and effectively contribute code to its development, and new contributors might find it too costly to join the project (Kohanski, 1998). Therefore, a theory of the open source software innovation process needs to explain how people sign up for the production of the public software good, under conditions where the cost of contributing vary. This paper intends to contribute to a theory of the open source software innovation process by examining joining behavior in a developer community.

Since the immanent puzzle of open source software innovation is the creation of the public good (Lerner and Tirole, 2002), research also has to uncover how joiners become *newcomers*, that is, how they make their initial contribution to software. In particular, it is important to understand what benefits newcomers derive from belonging to an existing developer community (e.g. Olson, 1965). According to the private-collective model (von Hippel and von Krogh, 2003), *newcomers* share with existing developers, greater benefits of revealing their innovations, than those outside the community (see also Callhoun, 1986; Taylor and Singleton, 1993). This is so because their ideas, bug reports, viewpoints, or code can be reviewed and commented upon by other developers and users, and in terms of learning benefits, the group's feedback can be direct and specific to the newcomer. Additionally, the formal acceptance of new code, the fixing of bugs or incorporation of feature requests results in direct benefits to the newcomer as their concerns regarding the software program now become the responsibility of the entire developer community.

The literature on commercial software development suggests that "modularization" (Baldwin and Clark, 2000) of the software code may increase a project's transparency, lower barriers to contribute, and allow for specialization by enabling efficient use of knowledge (Khoshgoftaar et al., 2001; Meyer and Seliger, 1998). Furthermore, efficiency in the innovation process requires that individuals specialize in certain areas of knowledge (Grant, 1996; Simon, 1991). Therefore, specialization in the developer community, by distinct software modules, could benefit the development process, and due to the barriers of understanding and contribution, this could be especially true for newcomers. Research is still lacking on the benefits of specialization in open source software innovation, and this paper attempts to contribute to a theory of the open source software innovation process, by uncovering if newcomers specialize and what may cause this specialization.

The lack of research on joining, contributing and specialization by newcomers, and the overall lack of a theory of the open source software innovation process, suggest a qualitative grounded approach to develop analytical categories and propositions (Glaser and Strauss, 1967; Meyers, 1997; Strauss and Corbin, 1990). We base our theory development on Freenet, an open source software project for decentralized and anonymous peer-to-peer electronic file publishing and retrieval over the Internet.

The paper is organized as follows; we first review the research method employed in our study (Section 2). Next we provide a history of Freenet and related development data (Section 3). We then proceed to theory induction (Section 4) by our analysis of joining and contributing behavior exhibited by those who became developers in Freenet. We conclude the paper by discussing implications of our study for theory and research (Section 5).

2. Research method

In this section we describe the research method employed in our study. Our research proceeded in three phases: sampling of case, data gathering, and data analysis. We selected one case in order to increase the depth of the analysis, acquire and report experience with the gathering of new and unfamiliar data (see Numagami, 1998).[2] Freenet was sampled for three reasons: firstly, in contrast to the Linux operating system (which is based on Unix) and most other open source projects, there is no pre-existing template

[2] Because the empirical setting of open source software innovation might be new and unfamiliar to many readers of *Research Policy*, we use a higher level of detail in our description of research methods than normal.

software architecture[3] for Freenet. Developers in the Freenet community are engaged in de novo design and development making it a radical innovation in distributed file sharing software systems (Oram, 2000). Consequently joiners should have to put in considerable effort before they can contribute, and newcomers should not be able to realize immediate benefits from specializing in module development according to an official and pre-defined modular software architecture. In contrast to projects emulating existing applications, Freenet contributors also do not know if their efforts will result in a suitable working product. Secondly, Freenet was launched not on the basis of workable code (see Lerner and Tirole, 2002), but on a master thesis in computer science written by its founder Ian Clarke outlining the theoretical principles of such a software system. This further compounds the practical considerations of writing code as most early developers will not have an initial software kernel to build upon (Raymond, 1999). Thirdly, Freenet is a young project in comparison to the more established OSS projects like the Linux operating system that has been in operation since 1991 or the Apache web server project which was founded in 1995. Our data covers the first year (2000) of the Freenet project, which was a critical phase in establishing sufficient momentum for the project by mobilizing newcomers. These three characteristics make Freenet a unique pilot case compared to the ones we know, with respect to joining, contributing and specialization of newcomers (Stake, 1995; Yin, 1994).

We gathered data from four different sources. Firstly, we conducted thirteen *telephone interviews*[4] in two rounds with eight Freenet developers identified from the developer list on the project's Internet homepage. Each interview took between 1 and 2 hours and was recorded and transcribed to facilitate easy data analysis. The rounds occurred between October 2000 to January 2001 and March to May 2001. All interviews were semi-structured with guidelines including developer background information, overall structure of the project, reason for joining and working on the project, specialization, and particular challenges in the project. In the first round of interviews basic understanding was gained of such factors as the technical characteristics of the project, critical events, and philosophy. Analysis of the first round of interviews indicated that one of the central issues of concern to the developers was the joining and specialization of newcomers. In the subsequent round we identified central as opposed to peripheral categories by obtaining information on the personal reasons for joining, how developers deal with joiners and others in the project, what type of contributions are made to projects, typical work-load, and specialization in the project. The interview guidelines provided a comparison across interviewees' arguments.

Secondly, we collected the project's public e-mail conversations stored in the projects' *mailing lists* which is archived on Freenet's website.[5] Since we focused on the contribution to technical development of Freenet, we gathered e-mail data from the 'development' list where the discussion centers on topics pertaining to the next release of Freenet, its design, emerging architecture and other technical aspects. The project also had three other mail lists pertaining to user support, technical discussions and announcements. Analysis of those lists showed them to be used infrequently during the timeframe of our study and thus not selected for analysis. This choice was also confirmed by our interviews of Freenet developers who indicated that the development list was the only place where any significant discussion took place. All the core developers also mentioned that the development list was the primary means of communication in the development team. Private e-mails to particular individuals or groups was not a standard practice for Freenet development and rarely occurred. We created a database of all messages including contributor identity, date and time for posting a message, mails responding to the message, and mail content for the period 1 January to 27 December 2000. The database included 11,210 single e-mail messages covering issues such as implementation details or code contributions. Similar to the method used by Mockus et al. (2002) each contributor to the discussion list was given a unique identification code based on an automated process of matching e-mail addresses.

[3] The architecture of software characterizes the functionality of specific modules within the software and the interdependencies and interactions among these.

[4] Eric von Hippel participated in the interviews of some of the developers.

[5] See at http://www.freenetproject.org.

We removed "junk" mail from the list and obvious repetitions of mails. Contributors with multiple e-mail identities were reconciled in one unique identity by the basis of manual examination of e-mail name field and signature in the text of the e-mail message which lead to a total number of 356 unique participants[6] on the development e-mail list.

The third source of data included the history of changes to the software code available via the project's software repository within the *Concurrent Versioning System* (CVS) source code management tool. CVS is a public 'version control' tool, designed to synchronize work and keep track of changes in the source code performed by developers working on the same set of files. CVS stores its version control information in a directory hierarchy on a central server, called 'the repository', which is separate from the user's working directory on his own personal computer. This allows developers to add or remove files easily, or to ask for versioning information of files. In addition to storing the project's source code, the CVS retains developers' comments regarding changes they have made to the source code CVS allows anybody to check out code from the repository. However, committing code, i.e. making changes to the source code, is restricted to those individuals that have been given permission by the administrator(s) of the CVS repository which in our case was the project founder and some early developers. Source code commits served as a major pool of data because progress of the project is reflected by the progress of source code modifications. The data retrieved from the period 1 January to 27 December 2000 included 1244 source code commits from 30 different developers. This comprised in total 54,000 lines of software code added, not counting the initial revisions of files.

Fourthly, in order obtain contextual understanding of the project we collected *publicly available documents* related to the project. Among the most important sources were the Freenet project web pages (e.g. the frequently asked questions (FAQ)[7]), Ian Clarke's master's thesis (1999), newspaper interviews with the core developers, and a technical paper (Clarke et al., 2000) describing the Freenet project written by some of the developers. In the case of doubt, ambiguity, or lack of data, clarification with developers in the project was obtained via e-mail or Internet chat.

The data analysis covered all 356 participants in the development list. We created four categories of participation in Freenet based on different roles played in the project. Members of the e-mail list (326 individuals) that did not submit any source code to the CVS repository were classified as "list participants" or non-developers. The remaining 30 individuals were assigned three temporal coding roles: (1) *Joiner*—someone who is on the e-mail list but does not have access to the CVS repository; (2) *Newcomer*—someone who has just begun to make changes to the CVS repository, and (3) *Developer*—someone who has moved beyond newcomer stage and is contributing code to the project.[8] Based on the development e-mail list database, we generated descriptive statistics on posting frequencies, the number of active participants over time and communication patterns among developers versus non-developers (list participants). This provided insight into the community size, 'life span' of developers and activity measures in order to identify patterns for joining. We then backtracked the evolution of the whole project and explored the developers' efforts of joining, contributing and specializing. These results were then used for in-depth qualitative content and process analysis in order to verify, complement, and extend the findings and our inductive theory building exercise.

We developed a coding scheme for joining (Strauss and Corbin, 1990), by analyzing e-mails on the development list, and coding the first e-mail (First Mail) of a prospective joiner, and the following e-mails (Subsequent Mail) from list participants, existing developers and the joiner, before a joiner was given CVS access. "First Mail" was based on message contents, and as well the responses from the community to the first mail (Response Mail). This provided

[6] Participants were numbered by an automated script according to the sender's email address. A manual analysis of the mails was performed in order to remove spam mails as well as to merge duplicate e-mail addresses. As the original numbering has not been changed, the 356 unique participants are not numbered sequentially and higher values are possible.

[7] http://freenet.sourceforge.net.

[8] No attempt was made to gather and analyze data on "lurkers"; passive but listening subscribers to the development mailing lists (Nonnecke and Preece, 2000).

a coding scheme of 31 different first mail message types, covering items such as bug fixes,[9] technical questions, and general comments on development, as well as 17 subsequent mail response types. We further grouped these into 14 broader items and analyzed their frequencies.[10] Two people separately coding the messages contributed to inter-coder reliability. Analysis of the "Subsequent Mail" was done for all 25 joiners until they were granted CVS access. The First Mail and Subsequent Mail of five developers was not available for examination as they had obtained CVS access and developer status prior to the start of the study period. The average number of messages until joiners became newcomers (23) served as a basis for examining the mails of all 326 list participants in the project who did not obtain CVS commit access. Confidence intervals were used to test whether the type of activity was significantly different from the two groups of people.

We analyzed developer activity over time by analyzing the frequency of source code commits,[11] as well as their specialization in different areas of the software code. Recall that, although seasoned developers might have tacit knowledge of Freenet's evolving architecture, no explicit model was available. In order to capture the type of source code modifications to identify specialization of newcomers and existing developers, we created a reference model of Freenet by examining all 1244 source code modifications and clustering them according to (a) the file structure within the code repository on the CVS and (b) the different tasks of the software. In order to enhance the validity of the reference model, we followed Jorgenson's (1989) advice by discussing and reviewing it in three iterations with developers, including the project founder.

[9] A bug is a programming defect or error that causes the software to malfunction.

[10] The full list of categories is available from the authors.

[11] A first attempt involved a cluster analysis of modified files by the each developer, but this was rejected as too imprecise for our purposes. A descriptive analysis of the number of touched files per developer indicated a high degree of specialization, but needed to be complemented by a qualitative analysis. In short, this first analysis did not tell us much about the distinctions between parts of Freenet critical to its purpose and performance, and those less critical.

3. Freenet history and development characteristics

In this section we provide a brief history of the Freenet project, its objectives and an overall characterization of the development process. The Freenet software enables a peer-to-peer network designed to allow for the distribution of information over the Internet in an efficient and anonymous manner. Ian Clarke started the Freenet project when he was a fourth year student at the University of Edinburgh, and completed the basic design in 1999. The overall design goals as stated on their development website are (Freenet.sourceforge.net, 2000); "Freenet is a large-scale peer-to-peer network which pools the power of member computers around the world to create a massive virtual information store open to anyone to freely publish or view information of all kinds. Freenet is: (1) Highly survivable: All internal processes are completely anonymized and decentralized across the global network, making it virtually impossible for an attacker to destroy information or take control of the system. (2) Private: Freenet makes it extremely difficult for anyone to spy on the information that you are viewing, publishing, or storing. (3) Secure: Information stored in Freenet is protected by strong cryptography against malicious tampering or counterfeiting. (4) Efficient: Freenet dynamically replicates and relocates information in response to demand to provide efficient service and minimal bandwidth usage regardless of load".

The first basic ideas and design of Freenet were outlined in Ian Clarke's master thesis (1999) entitled "A distributed decentralized information storage and retrieval system", which was published on an OSS community website (freshmeat.net) for people to comment on. He also initiated a project e-mail list and source code repository on SourceForge.net to coordinate the efforts of interested developers. Clarke was particularly interested in contributions that could turn these ideas and design into a workable software. The original document received limited attention, and Clarke sent an e-mail to the subscribers on the mailing list around Christmas 1999 announcing that he planned to step down as project leader due to personal reasons. Nobody volunteered to take over his position, but a number of people on Freenet's mailing list came to realize that there was a minimal amount of

Table 1
The reference model of the Freenet architecture[a]

#	Component name	Description
1	Routing	Which node to contact in order to request or insert data from or into Freenet
2	Data store	How data is stored on the local hard disk
3	Cryptography and security	Everything related to encryption
4	Keys	Keys represent a pseudo-unique ID for each file in Freenet and are used to identify data
5	Network joining	How new nodes can be hooked up into the Freenet network
6	Protocol and metadata	The internode protocol through which nodes are communicating
7	Clients	Clients are the interface which people use to communicate with their node and can fulfil several tasks
8	Client library	Commonly shared libraries among most clients to avoid duplicated code
9	Build and install	Scripts to compile the source code and to enable a convenient, user friendly installation of Freenet
10	Performance	Increasing the performance of the Freenet node
11	Search	How searching is accomplished
12	Testing and simulation	Code to test the functionality of Freenet and to simulate its behavior on a large scale
13	Documentation	Concerns all documentation which is written, like technical specifications, manuals and protocols
14	Node operation	Code which is required to get the Freenet server (node) up and running
15	GUINode & configurator	A graphical interface to the Freenet node and a graphical configurator (written in Java)
0	Miscellaneous	Everything not fitting into other components

[a] A more detailed description of the software architecture can be requested from the authors.

programming going on at the time. This mobilized efforts and in early 2000 several people made active contributions to Freenet.

Consistent with Freenet's philosophy of a widespread diffusion among users, the Java programming language was chosen as the development language for its ability to operate on heterogeneous computer platforms. Java also has strong network support facilities, a high integrated security level (e.g. against certain attacks) and it is said to be easier to debug than other computer languages like C or C++ (Freenet.sourceforge.net; Kohanski, 1998). However, Java also occasionally erected barriers for joiners. Many contributors to OSS projects have limited knowledge of Java and are confronted with the need to learn the programming language before contributing (Emurian et al., 2000).

Our reference model (Table 1 and Fig. 1[12] of the model) revealed that the Freenet software architecture evolved via 16 main modules. We assigned developers' efforts to the various parts of the architecture and discerned patterns of contributions and specialization form their activity in the model. On an even more detailed level of analysis, 53 'features' were identified as subcategories of these modules, that serve different tasks for the function of the module, however for the sake of simplicity we will limit our analysis and discussion to the 16 main modules.

The first beta[13] release of the Freenet software occurred in March 2000. Table 2 indicates that the Freenet developers followed Raymond's (1999) dictum of "release early, release often", with nine official

[12] Analyzing data with such a reference model deviates from previous methods. Koch and Schneider (2000) accessed publicly available data on the GNOME project in the CVS repository and in discussion groups. Their variables included demographic data of developers, and the project's productivity measured as lines of code (LOC) per hour, lines of code added or deleted, or number of postings to the mailing list. However, the validity of their "productivity" construct could be compromised because the metrics and results were not verified with developers (Glaser and Strauss, 1967). LOC is generally a weak proxy for a software development project's productivity (e.g. Caban et al., 2001; Rauscher and Smith, 1995), since it does not inform if code or a file is critical to the software's functionality. A very high number of LOC added does not necessarily mean a higher quality of code (e.g. bloated software). The "productivity" of 30 lines of a user manual file might not compare well with 30 lines introducing a new encryption algorithm. Besides LOC fails with binary documents (images, WinWord, compiled programs). In addition, since the source code also includes the contributors' comments to the code, LOC additionally distorts the projects' quantitative changes (see also discussions in Koch and Schneider, 2000; Humphrey, 1995). Being tested with the field, and by identifying and distinguishing central modules of the architecture, the use of a reference model remedies these weaknesses.

[13] A release that has working software code but may have many defects and many features missing.

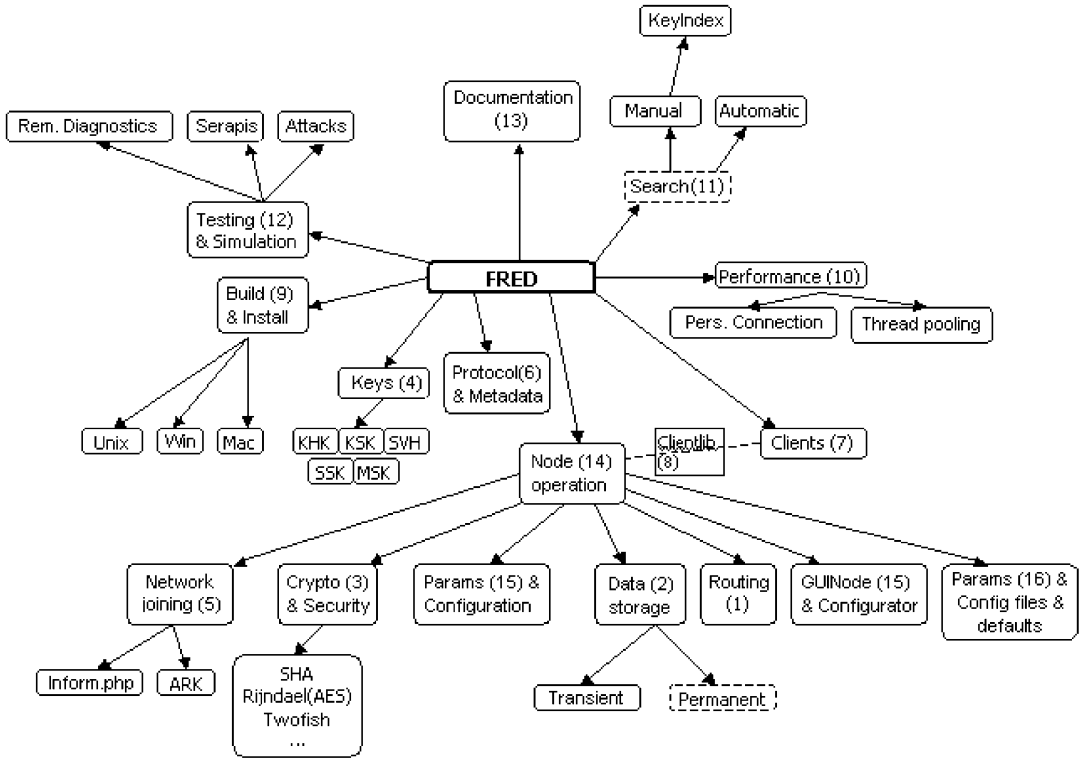


Fig. 1. The Freenet reference model—graphical overview.

code releases during our study period. Table 2 also shows there was a considerable amount of interest in the project with over 650,000[14] copies of the software downloaded by users over the Internet.[15] The download statistics indicate a large potential base of developers available to be mobilized for Freenet, since users are the primary source of developers in open source software projects (Moody, 2001; Raymond, 1999; von Hippel, 2001).

The development in a project like Freenet entails intense discussions on the software development e-mail list and the ongoing authoring and submission of source code as shown by changes made to the CVS repository. Overall, the project is characterized by temporary efforts, i.e. there is a high turnover in the developer community, confirming the findings from interviews that there is a need for understanding joining and contribution of newcomers and those that eventually become a developer. Fig. 2 shows the overall number participants on the developer e-mail

Table 2
Release dates and download numbers

Release	Release date	Week	Downloads
0.1 beta	9 April 2000	15	236573
0.2	1 May 2000	18	203562
0.3	17 September 2000	38	2191
0.3.1	19 September 2000	38	56595
0.3.2	1 October 2000	40	24245
0.3.3	9 October 2000	41	26674
0.3.4	31 October 2000	44	33660
0.3.5	18 November 2000	47	26827
0.3.6	24 December 2000	52	42461
Accumulated downloads			652788

[14] This number shows the cumulative number of downloads for all releases, the point release statistic, i.e. 42,461 downloads for release 0.3.6 provides an estimate of the user base for the project.
[15] Data for 2001 indicated a similar level of interest amongst users with an additional 618,000 downloads.

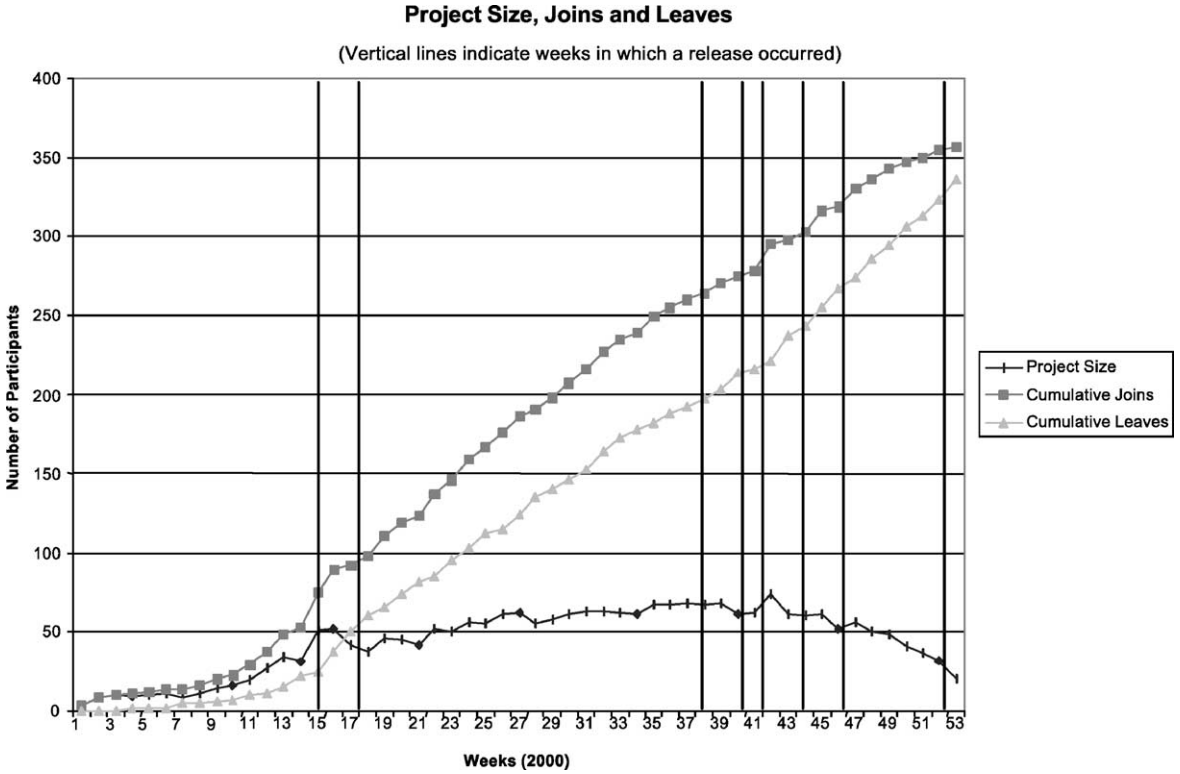


Fig. 2. Project size based on e-mail activity.

list over the year and the number of people entering and leaving[16] the list.

On average the project consisted of 45 (S.D. = 21) active participants per week. This number was achieved and stabilized around the first public release date week 15. As mentioned earlier, 356 unique individuals participated in the Freenet developer discussion list. They generated 1714 message threads consisting of 11,210 e-mail messages. A message thread is given by responses to an initial e-mail, covering a specific subject such as code which is required to get a Freenet node up and running, what is needed to enable a convenient, user friendly installation of Freenet on a user's computer and any other development related topic. The mean number of messages per thread was 6.5 (S.D. = 10.4). Significance testing indicated no difference in the means for messages per threads initiated by developers (regardless of temporal role) and list participants ($P = 0.11$, $t = 1.56$).[17] This indicates that there was an active community of participants and status in the developer community (i.e. who has CVS access) did not impact the day-to-day technical conversations. Developers and list participants were equally likely to be responded to in an initial e-mail. Fig. 3 shows the weekly distribution of e-mail messages and message threads over the year. On a weekly basis there were an average of 211 (S.D. = 106.1) messages and 32.34 (S.D. = 16.75) new message

[16] A join was recorded when a new participant posted their first message to the e-mail list. Content analysis of the messages showed that participants never gave notice of leaving, rather, they just stopped posting. Thus a leave was recorded based on the date of the last message of a participant. For each participant, we examined the forward weeks of e-mails to determine their leave date. Our data may suffer from "right censoring" effects as we did not have information on developers that posted initially in 2000 and then also continued participation in 2001.

[17] Twenty-six percent of threads contained only one message with no further public response from the developer list. Fifty-six percent of those non-response threads were initiated by 21 developers and the remaining were initiated by 118 other participants.

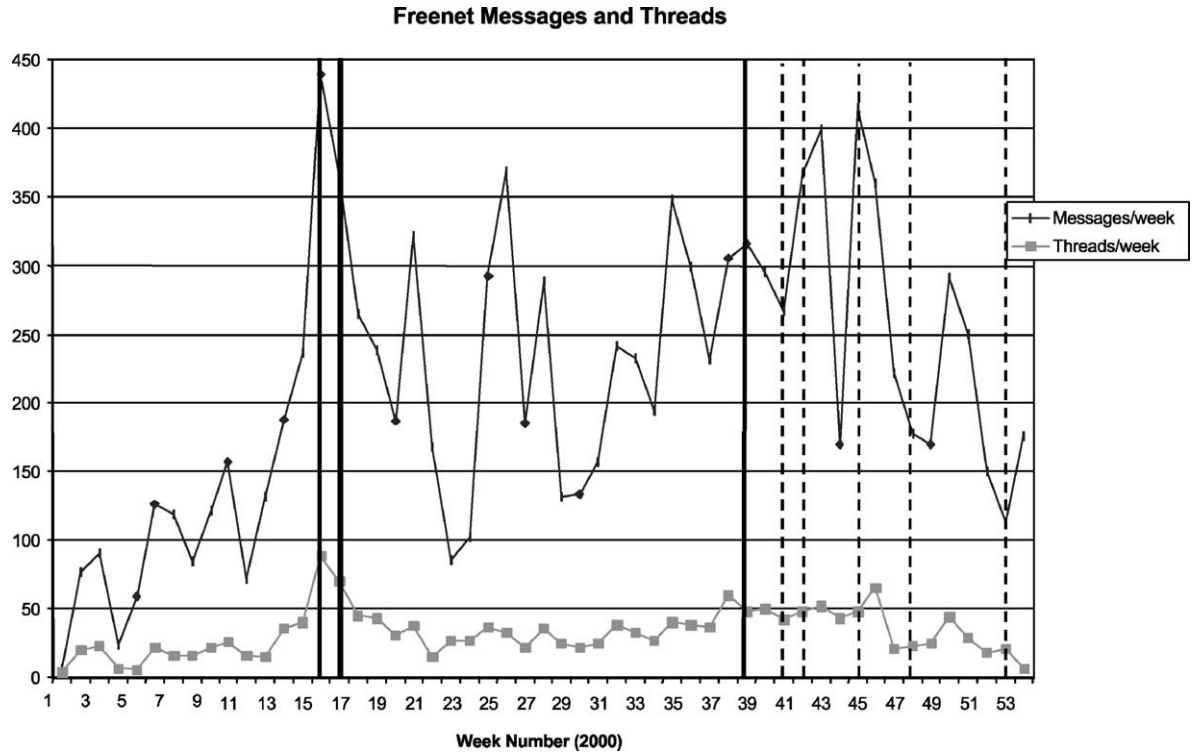


Fig. 3. E-mail messages and threads per week.

threads initiated. The solid lines on the graph indicate weeks where a major new release of Freenet was announced. The dashed lines indicate weeks where minor releases were announced to the development list. According to Table 2 the first release of Freenet (0.1 beta) was made in week 15. This resulted in a steep increase in e-mail messages, and as shown in Fig. 3, it sparked the interest and entry of new participants on the mailing list. Although the Freenet project initially did not start with a working code base, the presence of code, however early or defective, mobilized new contributors (Raymond, 1999; Lerner and Tirole, 2002).

A high 78% of the population of development list participants attempted to initiate dialogue, via starting a new thread, at least once. Of these attempts only 29 (10.5%) participants did not receive any reply to their initial posting and subsequently did not appear on the developer list again.

The character of the discussion activity shows that the behavior of the Freenet community resonates that of other open source communities (see e.g. Herrman et al., 2003). A "critical mass" of contributors sustained the development in the project. Participation in the development list was highly concentrated with four individuals, all of them developers, or 1.1% of the population accounting for 50% of the e-mail list traffic. The GINI coefficient for message authorship was 0.89 confirming this concentration of activity.

Fig. 4 shows the pattern of code commits to the project CVS repository over the year. On average there were 24 (S.D. = 18) commits per week.

Altogether 30 individuals (8.4% of total e-mail list population) had CVS commit access[18] for the project. There was a high degree of concentration in the code writing task with four developers (13%) making 53% of the code commits to the CVS repository (GINI

[18] The project did not normally have a separation between CVS commit access and writing code for the project, i.e. those writing code for the project would send their code to a specific person who had CVS commit access. Our analysis revealed only one e-mail list participant who modified the source code and then asked a developer with CVS access to commit the code in their name.

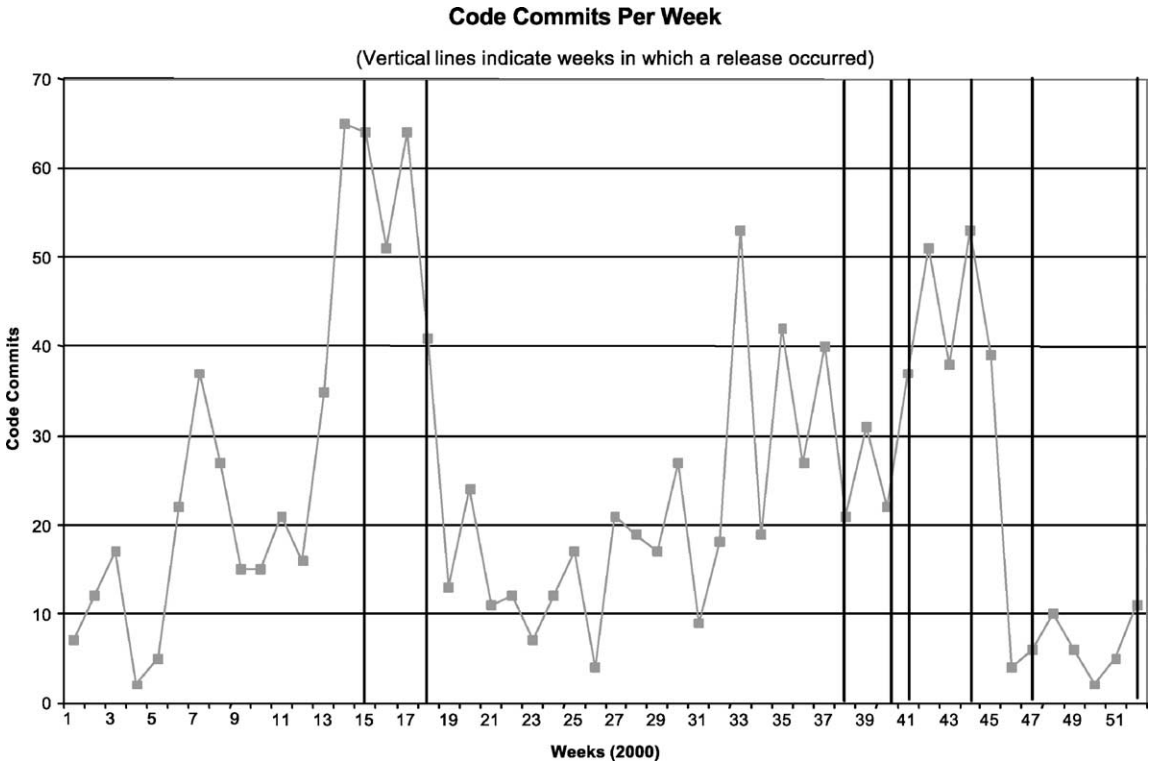


Fig. 4. Number of code commits per week.

coefficient = 0.77). Code commits culminated shortly before and remained for 2 weeks after the first public release (0.1 beta) took place in week 15. As developer #389[19] commented:

> A public release always leads to increased testing by new users, which in turn leads to the discovery of new bugs in the software, and commit of debugged code.

In summary we observe that although many people participate in the development discussion e-mail list for Freenet, code writing task is concentrated in a relatively smaller set of individuals who become developers for the project. Understanding the mechanisms by which these developers join the project and the areas in which they contribute and specialize are crucial elements towards developing a theory of the open source software innovation process.

[19] The developer number is higher than the total number of developers reported because some participants would post via multiple e-mail accounts. We manually combined those participants that exhibited such behavior. However we collapsed the coding such that the e-mail address and the corresponding unique ID associated with the highest number of posts was used as default.

4. Theory induction

In this section we develop propositions towards a theory of open source software innovation process. The propositions are grounded in observed behavioral strategies of newcomers attempting to join the developer group and their choices of the technical areas within the existing software code where they contribute.

4.1. Joining

Scholars have observed that newcomers joining technical projects must demonstrate some level of technical expertise as well as understanding of what the community expects in terms of behavior, in order

to make a contribution to technical development (Lovgren and Racer, 2000; Wenger, 1998). In studying the formation of collective action, Tilly (1999) underscored that "Joining" is a behavioral script that provides a structure for the activity of becoming a member of a collective action project. In our context, a joiner is a person who is eventually given CVS access, and thus becomes member of the developer community. Joiners emerge from the much larger group of list participants in the development e-mail list. We define a *joining script* as the level and type of activity a joiner goes through to become a member of the developer community. And therefore, joining scripts represents a cost to any would-be developer in the project.

4.1.1. Level of activity

One dimension of the joining script is the level of activity, that is the intensity of effort until a joiner is granted developer's status. Messages posted on the development mailing list and interviews with core developers indicated that often a significant period of observation (lurking), ranging from a couple of weeks to several months, was needed before someone felt they could contribute to the technical discussion:

> *Participant* #83: I've been lurking on this list for a while, so I thought I'd share some ideas about this [. . .].

> *Developer* #187: [I've] been lurking on freenet-dev for a while, but this is my first post to freenet-dev [. . .].

> *Participant* #78: 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'm a software developer with about three years of experience in the commercial world writing Java (1.1 and 1.2) 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 the 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!

Since the lurking period was unobserved in our study, the level of activity of joiners was measured in terms of the number mails to the developers list, prior to them getting access to the CVS repository. In the case of Freenet the average value of mails needed before a joiner became a developer was 23.4. However, the standard deviation was quite high (S.D. = 37.8) due to the fact that one of the 25[20] joiners generated 194 mails before being granted CVS access. Average time between first post and first commit on CVS was 40.8 days. Other participants over the whole year on average contributed only 9.8 mails (S.D. = 21.6).

4.1.2. Type of activity

The type of activity is also central to a joining script. In a study of software development, Glass et al. (1992) found differences in activities ranging from clerical to highly intellectual, complex and time consuming. Inspired by this study, we searched for differences in activities by proceeding in two stages: First Mail, and the type of activity a joiner underwent before being granted developer status (Subsequent Mail). Analyzing the First Mail of all participants using the coding technique we identified 14 non-disjunctive first e-mail contact categories.[21]

In 10% of the cases, people made personal introductions indicating level of skills, without necessarily providing any evidence of mastery in coding. Most commonly, in 40% of the cases a first e-mail posted a message to an already ongoing technical discussion on the developer list. By joining in a technical discussion a new participant can learn about the specific technical challenges of Freenet and signal interest and knowledge to existing project developers. As developer (#297) confirmed:

> If you wanted to join an open-source project, the first thing you do is get on the mailing list.

However, no joiner started out by unsolicited 'new' technical suggestions, perhaps indicating that it might be wise to start out humbly and not to boldly announce "great ideas" for solving problems. In fact none of

[20] Recall that five developers had access to CVS prior to the start of our study (pre-2000), one of which was Ian Clarke, the project founder. Two others were also very active developers.

[21] We do not give the full details of this analysis here, but summarize some of the results. The complete analysis can be obtained by the authors.

the 12.3% who suggested technical solutions without accompanying software code in their first post were joiners. In 16.7% of the cases, joiners started by offering source code for a bug fix, or an additional feature, in the form of actual software code submission, in the evolving software architecture, whereas only 4.6% of all non-developers did the same (mostly the announcement of external client projects).

A critical category we found is "Express interest to join as a coder". In 16.7% of the cases, this worked as an element of a joining script and in 9.5% it did not. By coding the responses from the community of developers (16 items), we further analyzed unsuccessful joining. In 36.7% of all cases, new participants on the development list stated they would like to join as coders, but got no response. However, in 56.7% of the cases members of the community encouraged the new participants to find some part of the software architecture to work on that would match with their specialized knowledge. In only 16.7% of the cases new participants were both encouraged to join and given specific technical tasks to work on. Given that Freenet, at the time of our study, did not have a well-defined and transparent architecture, general encouragement for newcomers to find something to work on might not have made the joining script less costly.

Given an average message thread-length of 6.5, and some threads exceeding 50 messages, joiners can contribute to an ongoing discussion, which in turn is likely to provide them with a better knowledge of the emerging software architecture and technical issues facing the project. Overall, the community welcomed people who announced their interest and indicated their skill levels, but they expected new participants to find their own tasks to work on. As we reasoned above, because a significant level of knowledge is often needed for entry, of both the emerging architecture as well as programming (Java), we also observed that many of those who announce their interests never become developers.

Next, we performed an analysis of all 564 e-mails of joiners until they were granted CVS access (with an average of 23.4 mails (S.D. = 37.8) before they were given developer priviliges) and of 1189 mails of non-developers. The average number of joiners' mails was used as a base for the examination of the other contributors' mails, so only the first 23 mails or the total message volume if less than 23, were taken into account. Of the resulting 22 categories of mails (see Appendix A for a complete list with example mail excerpts), nine categories turned out to be significant in distinguishing joiners' from non-developers' type of activity. Table 3 provides an overview of these categories and the descriptive statistics on joiner and non-developer types of activities. The numbers represent the relative percentage frequency of the occurrence of specific activity categories in the examined mails.[22]

Table 3
List of differences between types of activity

Activity category	Joiner	List participant
Significantly different activities		
Freenet question	0.018	0.087
Offer bugfix (code)	0.048	0.014
General technical discussion	0.430	0.276
Report bug	0.096	0.033
Repeated interest to contribute	0.016	0.003
Usage feedback (no bug report)	0.014	0.099
Request for resources (documentation, articles)	0.000	0.014
Request for help (to get Freenet running)	0.000	0.022
Point to technical resources/ refer to other projects	0.000	0.043
Not significantly different activities		
Express interest to contribute	0.053	0.075
Suggestion for improvements	0.126	0.153
Propose/outline bugfix (no code)	0.025	0.008
Coordination and organisation discussion	0.028	0.028
User support	0.014	0.017
Answer (technical) Freenet question	0.032	0.016
Self introduction	0.007	0.014
Announcing "external" contribution	0.044	0.047
Point out theoretical weaknessess of FN	0.018	0.007
Give Feedback on others contribution	0.011	0.010
Ask for a task to work on	0.005	0.003
Off topic discussion	0.082	0.072
Discuss legal/philosophical implications/matters	0.014	0.015

There are differences in behavior of joiners and non-developers. Non-developers ask more general

[22] A clarifying example: values of 0.5 respectively 0.3 for joiners and list contributors would mean the 50% of all joiners' mails and 30% of all other contributors' mails fall into that specific category.

questions about Freenet, and more frequently request help to get the software up and running on their own computer. They would also more frequently request resources such as documentation and articles. The interviews revealed that a joiner would typically lurk silently on the developer list and learn as much as possible in order to make a technical contribution, rather than entering into the development list asking general questions. Having learned about the technical details of the project, joiners would contribute more actively, than other contributors, to an ongoing technical discussion as a way of increasing their recognition. A developer (#334) stated:

> Actually, I just started getting involved. There would be a discussion about something, and I would just throw in my two cents about it. After a while, I was contributing to the discussion so much that everybody knew who I was and what I was doing there.

Joiners would more frequently report technical bugs in the software than other contributors, making developers aware of important "construction sites" in the project (see Kohanski, 2000). However, non-developers would give more general user feedback, without specific bug reports, than joiners. These reports were focussed on the use experience and often contained new feature requests.

Interestingly, whereas joiners in their first e-mail rarely indicated an interest to join the project, during the period that followed, they would more frequently than other contributors, repeat their interest to become a developer. The interview with developer #405 provides further interpretation of these findings:

> There are probably three kinds of people. One is, 'I started working with it. I saw these problems. I fixed them. Here they are.' That person gets in. There is the person who says, 'I am a JAVA engineer from Dallas, Texas. I've been working for five years, and I really would like to help. Give me something to do.' That person tends not to do anything. They tend to volunteer some expertise that doesn't get exploited yet. [. . .] The third type of person is the visionary, who says, 'I think Freenet is great, but it needs permanent storage, announcements, and broadcasting.' They tend to start fighting with the core architects, if they're lucky, who are actually making the decisions not to do that. Usually, they fight with lower-down people on the totem pole who are given the party line. They tend to never get in.

The quote indicates that there are implicit, but nevertheless important joining scripts in the Freenet project (no script has been written down). It shows the developer favors hand-on solutions to technical problem, and that demonstration of technical knowledge in the form of software code submission matters more than signaling of interest and experience.

The categories "technical discussions" also covers technical activities and reflects a joiners' technical and computational knowledge (e.g. Sonnentag, 1995). Either a message contains written software code, and/or it provides a technical contribution to the design and implementation of Freenet. Hence a joining script of Freenet is to undertake technical activities (Lovgren and Racer, 2000), providing concrete example to the community of a joiner's technical and computational knowledge (See Sauer et al., 2000).

Based on the data, analysis, and foregoing discussion, we conclude with the following proposition:

Proposition 1. *Participants behaving according to a joining script* (*level and type of activity*) *are more likely to be granted access to the developer community than those participants that do not follow the project's joining script*.

4.2. Contributing

In our analysis of Freenet, the transition from "joiner" to "newcomer" occurs when the joiner is given CVS access and makes a first contribution to the software. As reasoned above, there are private and collective benefits pertaining to community membership discussed in the literature (von Hippel and von Krogh, 2003; von Krogh, 2002; Wenger, 1998), but as well, there might be private and collective benefits resulting from specialization and division of labor in a project.

4.2.1. Specialization

In order to examine specialization of newcomers, we tracked the overall specialization in the developer community. The "specialization" construct is measured by the address of code submissions, i.e. to which

module of the software a particular contribution is made. "High" specialization indicates that the same modules within the code base were changed over time by a developer, while "generalization" indicates multiple modules were changed by a developer. The construct's dimension is the accumulated number of modules a developer created or changed over time. As we described above, modularity in the Freenet project was not explicit or visible. Our construction of the modules in the reference model (see Fig. 1 and Table 1) was based on an ex-post examination of source code areas and interviews with developers. However, this lack of explicit modularity did not deter specialization by modules as indicated by developer #4:

> I think we recently have specialized. Because [. . .] there are people like [developer #6] and they are working most with the client. And then [developer #101] and I have been doing the core stuff. And even with that, [developer #101] has been doing most of [. . .] the actual cryptography modules. [. . .] There is a guy who is working on graphical user interfaces.

Applying the reference model for the analysis of code commits to modules, we found evidence of high specialization. On average developers contributed to 4.6 (S.D. = 4.1) modules. However, 43% of all developers only contributed to up to two modules.

A similar high specialization can be shown for each of the 666 source code files:[23] Roughly 80% of all files were created and/or modified by a maximum of two developers during the period of analysis, with a mean value of 1.88 contributors per file. This finding matches astonishingly well with the research of Koch and Schneider (2002) who found a mean value of 1.8 contributors per file in the Gnome project.[24] Ian Clarke comments:

> It's . . . like, "I'll do this and I'll do this." Basically people throw ideas onto the mailing list and then people say: "I'll do that . . . I'll do that . . . I'll do that." You've got all of these tasks floating around and then people take tasks that they want to do. I think what makes the core programmers special is that they're willing to do the tasks that no one else does because I think they take a greater sense of personal responsibility and a more long-term approach. So some of the people who are not so active in the development will take the easy or the interesting stuff where the core developers are willing to do anything that needs to be done in order to further the progress of the project.

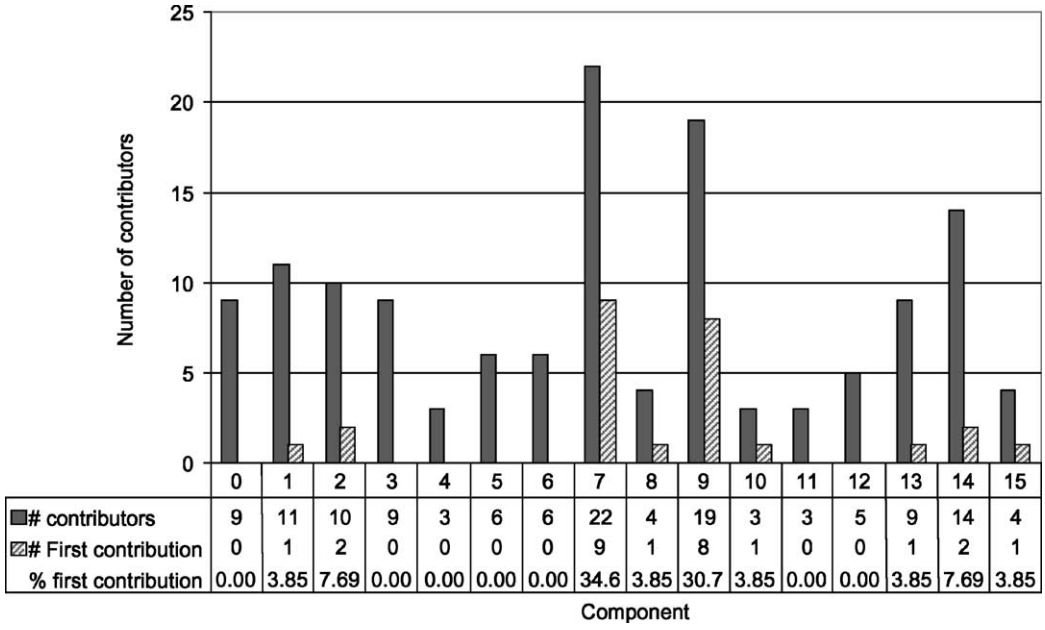
This quote supports the notion that developers specialize in coding modules because it is personally rewarding (von Hippel and von Krogh, 2003); they can apply their domain knowledge to modules and features in the emerging software architecture at low cost. Yet, three developers showed "generalization" or low specialization by contributing broadly to more than 13 modules. These are what Ian Clarke refers to as "core developers" with a long-term commitment to the project. The following quote from an interview with developer #101 provides a further interpretation:

> . . . well one of the problems we have right now that we are working on is cryptography (Module 3). We are adding a public key to the cryptography to the entire system, and unfortunately, any change you make in that affects just not only the protocol, which is what I am working on right now, but it affects how the keys are handled (Module 4), how the client interprets the keys (Module 8), how data is verified. Basically, that little change affects pretty much everything in Freenet and, therefore, the kind of people making those changes, myself and (developer #6) mainly, have to understand everything that happens in Freenet in order to do it.

As the emerging software architecture becomes increasingly complex, modules and features increasingly intertwined, some developers need to spread their efforts more broadly across the software. While high specialization allows for efficiency in innovation, there are also benefits to rotating people among jobs, in particular to broaden the understanding of a project, the increased sensitivity to coupling of tasks, and better management of interfaces (Cosgel and Miceli, 1999; Lindbeck and Snower, 2000). A small set of core developers therefore derived utility from working on many modules and ensured integration across the project as changes were made to modules that had other dependent modules.

In terms of newcomers' specialization, Fig. 5 shows the total number of developers that contribute to each module in the reference model, and how this related

[23] As of 31 December 2000. This number increased constantly over time.
[24] http://gnome.org.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# contributors	9	11	10	9	3	6	6	22	4	19	3	3	5	9	14	4
# First contribution	0	1	2	0	0	0	0	9	1	8	1	0	0	1	2	1
% first contribution	0.00	3.85	7.69	0.00	0.00	0.00	0.00	34.6	3.85	30.7	3.85	0.00	0.00	3.85	7.69	3.85

Component

Fig. 5. Number of contributors per component and where they contribute on joining.

to newcomer's contributions. This comparison is done across the population of newcomers making the first commits, rather than across time for each developer.

There were 30 contributors during the examined timeframe, so that, e.g. a value of 22 in Module 7 (clients) means that 73% of all contributors have changed the component "clients" at least once. The "first contribution" bars shows to which module new developers contributed during their first day. The base number for this figure is 26, since four out of the total of 30 contributors had already been developers prior to the timeframe we examined. Two modules are clearly predominating for joining behavior and shows high specialization. Sixty-five percent of new developers enter by either modifying the "build and install scripts" (Module 9) or by modifying existing clients or adding new clients (Module 7).

4.2.2. Contribution barrier

What can explain this high specialization of newcomers? Based on the interview findings, and the literature in commercial software development (e.g. Fichman and Kemerer, 1997; Kohanski, 1998), we propose a construct of "contribution barrier" erected by complex open source software technologies, where the following four items pertain:

1. ease of modifying and coding module (developer #101, #389);
2. the extent to which the potential developer can choose the computer language used to code for the module can vary (developer #389);
3. ease with which to "plug" the module into the architecture (developer #36);
4. the extent to which a module is intertwined or independently working from the main code (developer #36).

We illustrate the salience of the contribution barrier items by comparing all four dimensions for the 'build and install' module and the 'cryptography' module (see Table 1). As Fig. 5 indicates, 31% of all first contributions occurred in the build and install module (#9), whereas none of the first contributions occurred in the cryptography and security module (#3).

The first item, "Ease of modifying and coding" refers to the complexity of the source code and the level of difficulty of the used algorithms in order to achieve the desired goals. The level of 'difficulty' seems to be relative to the professional education and prior domain knowledge and expertise of the

developers. It varies across all developers and newcomers and cannot be adequately captured in a single global measure. Nevertheless it is possible to indicate some generic differences across modules. Using the two modules, cryptography and build and install, we found that the installer, which mainly performs simple tasks like copying files, uses much simpler routines than the cryptography module which contains complex algorithms and requires, e.g. in-depth knowledge of mathematics.

The second item concerns the computer language of a module. Some languages are complex and difficult to learn, while others, i.e. simple script languages[25] can be mastered fairly easily. Additionally, some computer languages are wide spread and can attract a large number of potential contributors, while others are known by few, and thus raise contribution barriers. Developers' personal preferences might be an additional reason why barriers to contribution is tied to language. One list participant stated, i.e. the dislike of having to use a proprietary software in order to be able to run the open source software:

> However, by principle, I will not install any Java product until the language becomes non-proprietary. I would like to start a C version of the server, ... (list participant #33).

In Freenet, the cryptography module is part of the core of the software architecture which written in Java, i.e. successful Freenet operation depends on this module, therefore requiring all cryptography algorithms to be written in Java as well. This poses a restriction on this module in contrast to, e.g. the build and install part, which is not part of the core and can be written in any programming language, allowing developers to use the language of their preference. We found at least five different types of programming languages used to code this module.

The third item of contribution barriers is the ease with which new modules can be 'plugged into' the existing software architecture. Clearly defined interfaces between the modules allow developers to use existing functionality without having to understand the rest of the specific algorithms used by Freenet. Also unwanted side effects are prevented by such an architecture, which allows to plug in modules easily, e.g. the build and install module with its functionality can be plugged into the main Freenet architecture without affecting it in any way. Also the build and install module uses predefined interfaces to communicate with the rest of Freenet, thus lowering the contribution barrier erected by the need for a detailed understanding on how Freenet itself works. The cryptography module stands in sharp contrast to the build and install module. Cryptography is closely intertwined with other modules in the architecture, thus requiring both a broad understanding of the overall functionality of Freenet, as well as a detailed understanding of more than one module. Two interview quotes illustrate how the ease with which to plug a module into the software architecture matters for the choice to contribute:

> Oh, well because at that time I felt I didn't know too much about technology. So, ... I didn't really want to mess too much with what was going on inside the node. So, on the user interface side, there's really no specialized knowledge (*of the emerging architecture—authors' addition*) needed. [...] And I saw that Freenet didn't have this and so that I could do something that I could just put in and it wouldn't be too controversial (developer #36).

> Yes. I think that what he [Ian Clarke] is saying there is that the node itself is so complicated and, unfortunately, so inter-tangled that it's not easily modularized and to do development on. Freenet, at least now, requires an understanding of fully everything that happens in the Freenet node. So that's why I think there are so few developers that actually work on the core node right now. There's a lot of learning curve there (developer #101).

Developer #36 mentions the relative benefit of specializing in the user interface, build and install, before more knowledge could be gained about the emerging software architecture. Developer #101 contrasts this with the core node functionality, including cryptography, where the "learning curve" for newcomers is very high because it requires thorough understanding of modules and features and their interconnectedness. As we reasoned, those modules are highly intertwined and specific to the project and require significant past investment in learning about the architecture, thus erecting contribution barriers for newcomers.

[25] A script is a list of commands to the computer that can be executed without the interaction of a user. A scripting language is a very simple computer language used to write scripts.

Table 4
Feature gifts made by newcomers during their first week

Developer	Module	Feature gift(s)
9	Cryptography and security	Test to ensure that files are properly encrypted
101	Cryptography and security	Three algorithms for encrypting files and better random number generator
101	Performance	Thread pool system for parallel processing
101	Testing and simulation	System to model overall Freenet network response
296	Cryptography and security	Safe operation while using web browser
297	Clients	Graphical interface for client
345	Clients	Address consistency in web browser
351	GUINode & configurator	Graphical interface for operation of Freenet server
389	Build and install	Microsoft Windows operating system build and install

The fourth item of contribution barriers concerns the extent to which a module is intertwined or independently working from the main code. Modules that work independently from the Freenet architecture can be used optionally and/or alternatively, and breaking them does not prevent the whole system from working. Such modules can be added or removed at any point, often without having to recompile the whole source code. For example, the build and install module is optional, because the Freenet software can also be installed and run without a proper installation utility. Developers consider the barriers for contributing to such modules to be lower because they do not risk breaking the whole system. This is not the case with all components. If the cryptography module is broken, Freenet will cease to work and might compromise the security and anonymity of its users.

The data, foregoing analysis and discussion leads us to formulate:

Proposition 2. *In an evolving software architecture of open source software projects, contribution barriers of modules* (*modifying and coding, variation in computer language, plug-in, and independence*) *are related to the specialization of newcomers*.

4.2.3. Feature gifts

We further analyzed instances where newcomers had provided whole modules or features (sub modules) to the software, rather than contributing to existing modules and features. The construct "Feature gift" is measured by whether the first contribution is an extension or feature in the reference model. Table 4 shows that there were nine such features added during the first week of a newcomers participation in the Freenet developer community.

In social exchange one can assume that individuals form relationships to maximize rewards and minimize costs, and that gifts are a part of this process. Our findings and analysis confirm an evolving idea in the literature; open source software innovation hinges on contributors giving gifts in the form of code (Raymond, 1999),[26] in this case features. Bergquist and Ljungber (2001) suggest that the gift giving is an important mechanism for organizing relationships between people. In their view, the open source software innovation process hinges on gift giving as a way of getting new ideas and prototypes out into circulation, and test their quality. In addition to the learning benefits obtained by contribution (von Hippel and von Krogh, 2003), in Freenet we found gift giving to increase early specialization of newcomers. The nine features given allowed newcomers to make a specialized contribution from the outset, and also allowed for specialization among other contributors. Hence we formulate:

Proposition 3. *Feature gifts by newcomers are related to their specialization in open source software projects*.

An important element of the feature gift giving was that the cost of creating and giving the gift was relatively low to the newcomers. Our interviews with the developers revealed that those that had contributed feature gifts did so on the basis of prior knowledge

[26] Eckstein (2001) notes that gift giving can be based on voluntary action in a community where norms and values give rise to such behavior. However, our data are weak on the social norms of the community, and we can only speculate that gift giving is related to the evolving sense of collective fate and obligation among developers.

and experience they had acquired and refined in other circumstances. As developer #36 told us:

> I guess (developer #101) had some thread-pooling code that he'd previously written, which was just lying around, essentially. And he said, "Freenet needs thread pooling," so he just sort of imported that whole stuff in.

In these circumstances, developers simply modified ready-at-hand code they had developed and used for other purposes, to the Freenet framework and then submitted their feature gift. The early feature gift is thus a way to quickly contribute code to the project. This finding is consistent with studies of the innovation process where pre-existing domain knowledge or direct experience is a source of new product ideas (Luthje et al., 2002; von Krogh et al., 2000) and a source of user-to-user technical support (Lakhani and von Hippel, 2003). This leads us to posit:

Proposition 4. *In open source software projects, feature gifts by newcomers emerge from the newcomers' prior domain knowledge and user experience.*

Our findings and analysis of feature gifts also shows another point related to contribution barriers. The Windows installer, a gift given by newcomer #389 to the build and install module, allowed other newcomers to make a fruitful contribution to Freenet in the area of Windows programming, although they may not have had intimate understanding of the evolving architecture of Freenet and perhaps lacked proficiency in Java programming. This gift seemed to have significantly lowered the contribution barriers of those that came after (see Proposition 3 and Fig. 5). Hence, we formulate:

Proposition 5. *Feature gift by newcomers are related to contribution barriers in an open source software project.*

5. Conclusion and implications

We studied joining and early contribution to the collective action of open source software innovation. Using data from Freenet, we inductively generated theory on the phases of joining a developer community and making the initial contributions to the software. In the first phase, we developed the construct of "joining script", and proposed that contributors who follow joining scripts in terms of level and type of activity are more likely to obtain access to the developer community. The transition from joiner to newcomer is achieved when a person is granted such access to the developer community, and to a privileged source code commit regime. We developed the constructs of specialization, contribution barriers, and feature gifts. We proposed that newcomers derive benefits from specializing in their contributions, that specialization of newcomers will be related to the contribution barriers in the project, that feature gifts given by newcomers will be related to their specialization in the project. Additionally feature gifts will be based on the newcomers prior direct experience and are related to contribution barriers and they create new entry points for developers who follow.

Some limitations confront our research. Firstly, the constructs and propositions are developed based on data from one case only. Although care was exercised to make the categories non-disjunctive and the constructs operational, measures, items, and the external validity of our propositions must be verified across a wider sample of cases. Secondly, although we could refine and eventually verify the reference model with an alternative field, its validity could be compromised by the allocation of files to the model. Research on projects with overt software architectures will not be subject to this limitation, but probably researchers could loose the benefits ensuing from studying a young project where mobilization of joiners and newcomers is critical. Thirdly, we studied an open source project where the commit regime was restricted to 30 people. Although similar regimes are known to hold for other projects, such as Linux, there may be projects that are fully open and where joining scripts are either more difficult or near costless. Fourthly, this research assumed that joiners expressing their intention to become a contributor actually intend to contribute to the project. Further research could reveal that for some reasons joiners frequently state their interest to join without real intentions to do so, which would bias the number of people not succeeding in becoming a developer.

The study has implications for research on open source and commercial software innovation. Firstly, it is important to recall that open source software development has characteristics of collective action aimed at producing a public good. This calls for an extended theory of innovation (von Hippel and von Krogh, 2003). Current theorizing builds on the premise that all or most innovation will be supported by private investment and that private returns can be appropriated from this (e.g. Demsetz, 1967). To encourage such investment, society grants innovators intellectual property protection (Grandstrand, 1999). The situation is different in open source where the protection mechanisms partly or fully guarantee the rights of the user, by sustaining free revealing of software code (Stallman, 1999). An extended theory must explain why, what, and how expert developers contribute for free to the production of a public good. The premise of such a theory, as shown by this study, should be that contributions are not costless and that significant costs are associated with joining a project. A joining script of a project implies significant levels of technical activities conferred upon joiners before they are granted the access to the commit regime. Future research should explain not only variance in joining scripts across a population of projects, but also the motives of joiners and how they change over time as they work their way into the project. This will help get a more complete picture of those factors enabling growth and continuation of projects.

Secondly, in line with existing theory, we found that there are community-related benefits available to newcomers (von Hippel and von Krogh, 2003; Raymond, 1999), but added that specialization in the project incur benefits for newcomers. More research should be devoted to test if the same patterns of newcomer specialization can be identified across a population of projects. Future studies should also investigate whether or not developers change their degree of specialization over time, as they "move down the learning curve" in the software architecture.

Thirdly, one might have the impression of open source software innovation as a market where contribution equals participation. If Freenet is representative, this is not so. On the one hand, the market for a technical solution might matter less for an entry into a developer community, than the knowledge and interest demonstrated through a sustained level of high quality technical activity.[27] On the other hand, some anecdotal information available in our data revealed that joining script relates to concerns in the community about software architecture, but the method did not give sufficient substance to further develop this line of inquiry. Theorists of collective action have suggested the concept of "non-redundant groups", in which the contributions of all members of a certain type is needed for the production of a specific public good (see Cortazar, 1997). The analysis of specialization in Freenet indicates the community could be redundant to some extent, but also that newcomers were attracted to those modules where entry points were present and visible and contribution barriers were low. The problem, therefore, is for the community as well as the joiner to understand whether the contributions of joiners is essential to the project, and also how this relates to the contribution barriers. Future research should attempt to uncover how the evolution of the software architecture changes joining scripts, so that for example, the joining script is less costly for people who demonstrate high level of technical knowledge in areas vital to the project, than for those joiners whose technical knowledge is significantly overlapping with that of the existing developer community.

Fourthly, a pressing issue of concern to researchers interested in both technical and social aspects of software innovation is whether or not there are linear or non-linear production relationships between developer input (team size) and the project's productivity. For example, in commercial software development Banker et al. (1994) found non-linear effects: there are both increasing and decreasing economies of scale. Future research should attempt to investigate the production relationships between developer input and productivity. There might be a critical size of a developer community, 30–40 people, that can produce open source software in an efficient manner (see Wayner, 2000), but we clearly need more research on factors, such as the costs and mechanisms of coordination impacting on critical size. Studies should compare the

[27] Bidault and Fischer (1994) made a similar observation regarding technology transactions between firms. Rather than buying a superior technology in the market, firms tend to engage in technology transactions with partners about which they possess information, that is, whose "identity" is known.

productivity of the private-collective model of innovation with traditional commercial software innovation.

Fifth, MacCormack et al. (2001) found evidence in commercial Internet software development that higher performance of the development project was associated with the use of development teams with greater knowledge from several releases of a particular software (generational experience). Open source software development is a voluntary activity, and as can be inferred from the analysis of development activity in Freenet, a high turn over among developers and other contributors may reduce generational experience. Intuitively, transparent development processes and software architecture could outweigh the loss of generational experience in producing high levels of a project's productivity (also by reducing contribution barriers), but this needs examination in further research. Research should investigate if projects differ much in terms of turnover among various contributors, and as well what factors impact on turnover. Moreover, the sharing of knowledge among ingoing, outgoing, and remaining developers needs more attention, and also how turnover impacts on the performance of an open source software project.

Solid theory building and empirical studies on the social aspects of software development is still lacking (Nambisan and Wilemon, 2000; Swanson, 1994). In commercial software projects the technical aspects of innovation are often obscured by lack of access to the source code. For this reason, Kemerer and Slaughter (1999) noted that it is methodologically challenging to capture incremental changes in the software architecture, and their causes. This study shows that the open source software development process is transparent, both with regards to the social and the technical aspects. As a research setting, an open source software project permits empirical work on innovation issues such as choices of design, architectural development and modularization, activities preceding and following software release, user feedback, quality improvement, social dynamics in innovation communities, integration of newcomers into a project, and leadership. Eventually, work on open source software development and its commercial counterpart will mutually benefit from exchange of results, and jointly they will contribute to an extended theory of private-collective innovation.

Acknowledgements

We are grateful to helpful comments from two anonymous reviewers. We also thank Chris Argyris, John Seely Brown, Eric von Hippel, Audris Mockus, Patrick McCormick, Luis Villa, Stefan Haefliger, Petra Kugler, Heike Bruch, Simon Gächter, Simon Peck, and Hari Tsoukas for helpful comments and suggestions. Ben Ho and Craig Lebowitz provided technical assistance with data importation and parsing. We would like to thank Ian Clarke and the Freenet developers for their willingness to participate in our study and providing key insights into the open source development process. Karim R. Lakhani would like to acknowledge the generous support of the Boston Consulting Group and Canada's Social Science and Humanities Research Council doctoral fellowship. Georg von Krogh and Sebastian Spaeth acknowledge the generous support from the Research Foundation at the University of St. Gallen.

Appendix A. List of activity categories with excerpts of a typical e-mail

A.1. Express interest to contribute

> Hi all, I'd like to contribute to the development of Freenet. I've found that the best way for me personally to understand the code is to document it. So I guess the first thing I'd like to do is go through all the code and document it (participant #253).

A.2. Freenet question

> How will people find a document when it is first uploaded into freenet? The only nodes that can find it are the ones that are closer to it's pocket in keyspace than to any other pocket. The document can't spread if nobody can find it (pockets must occur, because inserts are done to proximal keys, and pockets are a Good Thing anyways) (participant #7).

A.3. Suggestion for improvements

> I agree that adding special routing tables for every single situation is just asking for confusion. [. . .] I would like to propose a flexible plan that might get

around these issues. There is no doubt that there is some importance to a node knowing the "quality" of its surrounding neighbours. This could include ping time, node uptime, node cpu load, available bandwidth and could be passed back and forth to nodes through the handshaking mechanism (a mechanism which has amazing application potential for future freenet development . . . for all those people questioning "why do we bother with a handshake?"). This quality data could be stored in a single table mapping various Ips to their associated vitals. This data would be useful to many different parts of the code but here I would single in on routing . . . (participant #46).

A.4. Offer bugfix (code)

I hacked around the problem by putting private static class RequestAbortException extends Request.RequestAbortException {}in classes DataRequest and InsertRequest. Obviously a kaffe bug, though (participant #76).

A.5. General technical discussion

I know the planned hash function for CHK but by, e.g. modifying one word in a document I assume you could more likely get that same key with fatal consequences. Anyhow, the chances exist, in which case two different versions of one CHK in Freenet exist [and one would retrieve the wrong document].

The chances two insert two different docs with the same key despite the checking mechanism still exist, but are much smaller (I would think). So we shouldn't disable that feature that makes Freenet more reliable in terms of consistent keys (participant #389).

A.6. Propose/outline bugfix (no code)

Yes, that's it. If you run out of entropy reading /dev/random on FreeBSD, /dev/random stops returning bytes (i.e. the read() is short) until more entropy appears from the interrupt channels. Oops, and I don't have any interrupt channels but the keyboard . . . .

/dev/urandom always returns data; you might want to make it try to open that before /dev/random (participant #76).

A.7. Report bug

I am seeing evidence of message ID corruption, the following illustrates [. . .].

Mar 24, 2000 2:29:43 PM:client/RequestClient: Minor:The request got stuck on a broken node but has been restarted at a depth of 0.

Mar 24, 2000 2:37:13 PM:MessageFactory.java: Normal:Unknown messagetype-java.lang.ClassNotFoundException: Freenet.message.

I noticed in the code that the ID is handled as a long integer. Is there a type mismatch somewhere in the code? (participant #45).

A.8. Coordination and organisation discussion

I second this motion. It's so hard to read up on this list and try and follow all of the discussions: sometimes people have good points; other times people's points seem valid but have some flaws. It would be nice if at the conclusion of a discussion authors of ideas would write-up their proposals and posted them to the website. More specifically, I'm still uncertain about the conclusion of the following discussions:

(a) mechanism for updating documents;
(b) search and metadata;
(c) subspace partitioning (the language spec is a good start);
(d) exact protocol for trusting your neighbors and discovering rogues (participant #208).

A.9. User support

> I read Oskar's response to your problem and wasn't sure if he was saying that port forwarding would work or not.

Port forwarding works just fine. I ran Freenet behind an ISDN router (Zyxel 100IH) OK. This is set up to forward ports to a particular PC. I used a version of W2K, and did not have to do anything

special. If Netmeeting work for you, then Freenet should as well (participant #116).

A.10. Answer (technical) Freenet question

> What is the current status of metadata implementation in Freenet, and how do I get at this metadata from the command line client?

> Can I get a MIME content type? The -metadata [FILE] option will do it. I assume you're not using an output file. The client'll output something like this to stdout: [. . .] (participant #345).

A.11. Self introduction

My name is [XXX]. I live and work in Cheltenham, England. I work in a small computer shop called [XXX], fixing and building computers.

I study in a nearby city called Gloucester. Other than computers and Free Software my mail interests are the sciences, language (though not in the science that you might expect) and drama. Really I have know idea what I want to do with the rest of my life, but Brandons "company programming dynamic non-linear multi-user virtual reality worlds" sounds like the type of thing I'd hope to be doing! (participant #9).

A.12. Announcing "external" contribution

My pet project for this month—a freenet library and client in c—is now in a previewable state. There's still a lot of code cleanup and big changes to the client in store and the library still needs a lot of work. I've tested the client with Snapshot 4–27 and it worked. So take a look: XXX [removed URL to project] (participant #52).

A.13. Point out theoretical weaknessess of FN

I just want to point out one potential danger of this mapping idea. It would cause people who request data to have both a descriptive key and the actual message on their node whereas the routing of data would normally split the two up. So you get a record of what you request and one of the nice things about Freenet is that requested data and data routed through your node are indistinguishable (participant #19).

A.14. Repeated interest to contribute

Lads, would you be prepared to let me loose on it? I can do a nice job of it, and you're welcome to inspect my code before I check it back in. If you could decide now it would be good, because my evening is just beginning [. . .] (participant #297).

A.15. Give feedback on others contribution

> OK, I have added the code in these two places. Hopefully things will be better in tomorrow's snapshot. Unless of course we have to wait another week for people to start running the new code.

Nah, just applied your patches and restarted the server. I think you are truly the Orkin man:-) [. . .] (participant #45).

A.16. Ask for a task to work on

I have about 5 years coding experience in C, 3 in C++, and about 1.5 in Java. I can do Perl pretty well too. I have a spring break coming up, so maybe if someone could point me to a little part where I could contribute? (participant #56).

A.17. Usage feedback (no bug report)

I put up a node on piclab.com:800 and it seems to be working. InsertClient is a bit inelegant: it seems to have inserted the file correctly, but then timed out and killed itself. I don't yet know the code well enough to decide whether that's OK (participant #26).

A.18. Off topic

OK Since I'm on AOL I'm an EVIL and stupid [person] who doesn't know the first thing about computers, the Internet, or how big money works! . . . right? I don't like the way the corporate is shaping the world wide web (note not the Internet) for their

own, and only their own benefit [. . .]. I think capitalism is a good system except it is fatally flawed in that: once you have most or everything of monetary ($) value what else is there to have? there is control—your first and last goal (participant #314).

A.19. Request for resources (documentation, articles)

Hi, I was wondering if someone could be good enough to send a copy of Freenet.ps to XXX [removed e-mail address]; this is the paper "A Distributed Decentralised Information Storage and Retrieval System" by Ian Clarke (participant #7).

A.20. Request for help (to get freenet running)

Hi, I have just downloaded your program and I cant seem to get it to open. I saved it to desk top and along with the freenet software I also downloaded the Java software. I even unzipped both downloads and when I click on any of the folders in them it asks you to choose which program to use to open the program, and I'm not quite sure which to open it in. I tried it in explorer and it does'nt work. I'm very anxious to begin using your software, so please help me. Thank you (participant #325).

A.21. Discuss legal/philosophical implications/matters

However, it's clear to me (and many others) that Freenet is likely the singularly most interesting and powerful piece of software currently in development. So powerful, in fact, that I doubt that by the time the platform becomes quite feature-comprehensive, easy-to-use, and popular (I'd give ∼1–2 years for 5 million nodes?) that the governments of the world are going to do all they can to stop it, legally and technically.

Therefore, it seems to be prudent for someone to give a legal analysis [of Freenet]. (participant #37).

A.22. Point to technical resources/refer to other projects

Hmmm, All these talk of multicasting jogged my memory and what popped out was this article appearing on DDJ #312, May 2K, "Scalable Multicasting File Distribution".

Sounds familiar? You can get it at http://www.ddj.com/articles/2000/0005/0005i/0005i.htm. No doubt this was written by a researcher in MS (horrors!:), but was done quite professionally. There are code available but . . . in C/C++. Another article on general multicasting is http://www.ddj.com/articles/1997/9710/9710b/9710b.htm?topic=communications (participant #152).

References

Baldwin, C.Y., Clark, K.B., 2000. Design Rules, vol. 1. The Power of Modularity. MIT Press, Cambridge, MA.

Banker, R.D., Chang, H., Kemerer, C.F., 1994. Evidence of economies of scale in software development. Information and Software Technology 36 (5), 275–282.

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

Bidault, F., Fischer, W.A., 1994. Technology transactions—networks over markets. R&D Management 24 (4), 373–386.

Caban, A., Cimino, C., Swencionis, C., Ginsberg, M., Wylie-Rosett, J., 2001. Estimating software development costs for a patient multimedia education project. Journal of the American Medical Informatics Association 8 (2), 185–188.

Callhoun, C., 1986. The radicalism of tradition: community strength or venerable disguise and borrowed language? American Journal of Sociology 88 (6), 886–924.

Clarke, I., 1999. A distributed decentralised information storage and retrieval system, Unpublished Masters Thesis. University of Edinburgh, Edinburgh.

Clarke, I., Sandberg, O., Wiley, B., Hong, T.W., 2000. Freenet: a distributed anonymous information storage and retrieval system. In: Proceedings of the Paper Presented at the Designing Privacy Enhancing Technologies in International Workshop on Design Issues in Anonymity and Unobservability. Berkeley, CA.

Cortazar, R., 1997. Non-redundant groups, the assurance game, and the origin of collective action. Public Choice 92 (1–2), 41–53.

Cosgel, M.M., Miceli, T.J., 1999. Job rotation—cost, benefits and stylized facts. Journal of Institutional and Theoretical Economics 155 (2), 301–320.

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

Eckstein, S., 2001. Community as gift-giving: collective roots of volunteerism. American Sociological Review 66 (6), 829–851.

Emurian, H.H., Hu, X., Wang, J., Durham, A.G., 2000. Learning Java—a programmed instruction approach using applets. Computers in Human Behavior 16 (4), 395–422.

Fichman, R.G., Kemerer, D.F., 1997. The assimilation of software process innovations: an organizational learning perspective. Management Science 43 (10), 1345–1363.

Glaser, B., Strauss, A., 1967. The discovery of grounded theory: strategies for qualitative research. Aldine de Gruyter, New York, NY.

Glass, R.L., Vessey, I., Conger, S.A., 1992. Software tasks: intellectual or clerical. Information and Management 23 (4), 183–192.

Grandstrand, O., 1999. The Economics and Managment of Intellectual Property: Towards Intellectual Capitalism. Edward Elgar, Cheltenham.

Grant, R.M., 1996. Towards a knowledge-based theory of the firm. Strategic Management Journal 17, 109–123.

Herrman, S., Hertel, G., Niedner, S., 2003. Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux kernel. Research Policy (special issue on open source software development).

Humphrey, W.S., 1995. Discipline for Software Engineering. Addison-Wesley.

Jorgenson, D.L., 1989. Participant Observation: A Methodology for Human Studies. Sage, Newbury Park, CA.

Kemerer, C.F., Slaughter, S., 1999. An empirical approach to studying software evolution. IEEE Transactions on Software Engineering 25 (4), 493–509.

Khoshgoftaar, T.M., Allen, E.B., Jones, W.D., Hudepohl, J.P., 2001. Cost-benefit analysis of software quality models. Software Quality Journal 19 (1), 9–30.

Koch, S., Schneider, G., 2000. Results from software engineering research into open source development projects using dublic data. In: Hansen, H.R., Janko, W.H. (Eds.), Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft.

Koch, S., Schneider, G., 2002. Effort, cooperation and coordination in an open source software project: GNOME. Information Systems Journal 12 (1), 27–42.

Kohanski, D., 1998. Moths in the Machine. St. Martin's Press, New York, NY.

Kohanski, D., 2000. Moths in the Machine: The Power and Perils of Programming, 2nd Ed. St. Martin's Press, New York.

Lakhani, K., von Hippel, E., 2003. How open source software works: "Free" user-to-user assistance. Research Policy (forthcoming).

Lerner, J., Tirole, J., 2002. Some simple economics of open source. Journal of Industrial Economics 50 (2), 197–234.

Lindbeck, A., Snower, D.J., 2000. Multitask learning and the reorganization of work—from Tayloristic to Holistic organization. Journal of Labor Economics 18 (3), 353–376.

Lovgren, R.H., Racer, M.J., 2000. Group-dynamics in projects: don't forget the social aspects. Journal of Professional Issues in Engineering Education and Practice 126 (4), 156–165.

Luthje, C., Herstatt, C., von Hippel, E., 2002. The Dominant Role of "Local" Information in User Innovation: The Case of Mountain Biking. MIT Sloan School of Management Working Paper, #4377-02.

MacCormack, A., Verganti, R., Iansiti, 2001. Developing products on "Internet Time": the anatomy of flexible development process. Management Science 47(1), 133–150.

Meyers, J.D., 1997. Qualitative research in information-systems. MIS Quarterly 21 (2), 241–242.

Meyer, M.H., Seliger, R., 1998. Product platforms in software development. Sloan Management Review 40 (1), 61–74.

Mockus, A., Fielding, R., Herbsleb, J., 2002. Two case studies of open source software development: Apache and mozilla. ACM Transactions on Software Engineering and Methodology 11 (3), 1–38.

Moody, G., 2001. Rebel Code: Inside Linux and the Open Source Revolution. Perseus Press, New York.

Nambisan, S., Wilemon, D., 2000. Software development and new product development: potential for cross-domain knowledge sharing. IEEE Transactions on Engineering Management 47 (2), 211–220.

Nonneke, B., Preece, J., 2000. Lurker demographics: counting the silent. In: Proceedings of the SIGCHI conference on Human factors in computing systems. Association for Computing Machinery, ACM Press, New York, pp. 73–80.

Numagami, T., 1998. The infeasiblity of invariant laws in management studies: a reflective dialog in defense of case studies. Organization Science 9 (1), 2–15.

Olson, M., 1965. The Logic of Collective Action. Harvard University Press, Cambridge, MA.

Oram, A., 2000. Gnutella and Freenet Represent True Technological Innovation http://www.openp2p.com/pub/a/20805/12/2000.

Pliskin, N., Balaila, I., Kenigshtein, I., 1991. The knowledge contribution of engineers to software development: a case study. IEEE Transactions on Engineering Management 38 (4), 344–348.

Rauscher, T.G., Smith, P.G., 1995. Time-driven development of software in manufactured goods. Journal of Product Innovation Management 12 (3), 186–199.

Raymond, E., 1999. The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary. O'Reilly and Associates, Sebastapol, CA.

Sauer, C., Jeffrey, D.R., Land, L., Yetton, P., 2000. The effectiveness of software development technical reviews: a behaviorally motivated program of research. IEEE Transactions on Software Engineering 26 (1), 1–14.

Sawhney, M., Prandelli, E., 2000. Communities of creation: managing distributed innovation in turbulent markets. California Management Review 42 (4), 24–35.

Simon, H., 1991. Bounded rationality and organizational learning. Organization Science 2 (1), 125–134.

Sonnentag, S., 1995. Excellent software professionals: experience, work activities, and perceptions by peers. Behaviour & Information Technology 14, 289–299.

Stake, R.E., 1995. Case studies. In: Denzin, N.K., Lincoln, Y.S. (Eds.), Handbook of Qualitative Research. Sage, Thousand Oaks, CA, pp. 236–247.

Stallman, R., 1999. The GNU Operating System and the Free Software Movement. In: DiBona, C., Ockman, S., Stone, M. (Eds.), Open Sources: Voices from the Open Source Revolution. O'Reilly, Sebastapol, CA, pp. 53–70.

Strauss, A., Corbin, J., 1990. Basics of Qualitative Research. Sage, Thousand Oaks, CA.

Swanson, E.B., 1994. Information systems innovation among organizations. Management Science 40 (9), 1069–1092.

Taylor, M., Singleton, S., 1993. The communal resource: transaction cost and the solution to collective action problems. Politics and Society 21 (2), 195–214.

Tilly, C., 1999. Durable Inequality. University of California Press, Berkeley, CA.

von Hippel, E., 2001. Innovation by user communities: learning from open-source software. Sloan Management Review 42 (4), 82–86.

von Hippel, E., von Krogh, G., 2003. Open source software and the private-collective innovation model: issues for organization science. Organization Science 14 (2), 209–233.

von Krogh, G., Ichijo, K., Nonaka, I., 2000. Enabling Knowledge Creation. Oxford University Press, New York, NY.

von Krogh, G., 2002. The communal resource and information systems. Journal of Strategic Information Systems 11 (2), 85–107.

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

Wayner, P., 2000. Free For All: How Linux and the Free Software Movement Undercuts the High-Tech Titans. HarperBusiness, New York.

Wenger, E., 1998. Communities of Practice: Learning, Meaning and Identity. Cambridge University Press, Cambridge, UK.

Yin, R.K., 1994. Case Study Research: Design and Methods, second ed. Sage, Thousand Oaks, CA.