Keeping it going: The Everyday Practices of Open Source Software

Eric Monteiro*, Thomas Østerlie*, Knut H. Rolland*, Emil Røyrvik**

Norwegian Univ. of Science and Technology (NTNU),

***Dept. of Computer and Information Science, **Dept. of Social Anthropology,**

N-7491 Trondheim, Norway

***{ericm,thomasos,knutrr}@idi.ntnu.no, **emilr@svt.ntnu.no**

Abstract:

A key challenge in open source software (OSS) projects is to cultivate and nurture a motivated community of developers. More often than not, OSS projects struggle with a chronic lack of man-power, which ultimately threatens to undermine the whole project. This introduces a certain form of fragility to OSS projects. To avoid collapsing, the enabling of mechanisms that fosters continuity is of critical importance for keeping the project going. This does not take place through heroic, 'big' decisions and events, but is the life-blood of everyday practices in OSS projects. Through a case study of the OSS distribution Gentoo Linux we analyze three such types of mechanisms: organizational, ritual and technical. Methodologically, we emphasize everyday interactions by drawing predominantly on synchronous (IRC) communication. Through a close study of the everyday interactions of development and maintenance work in Gentoo Linux, we challenge the uniqueness of OSS, as expressed for instance in the gift economy metaphor, compared to other software development projects and distributed project work. We conclude by providing analytical and methodological implications for OSS studies, as well as practical implications for distributed,

knowledge based project work more in general.

Keywords: Open source software, knowledge management, interaction rituals, everyday practice

Introduction

- Not, then, men and their moments. Rather moments and their men.

E. Goffman

Open source software (OSS) has been studied within disciplines as diverse as software engineering (Mockus et al. 2002), law studies (Kesan and Shah 2002), and philosophy (Himanen 2001). There exist, in addition, more historical accounts of OSS communities (Moody 2002, Raymond 2001) and the hacker culture (Levy 1984). What is most relevant to our study, however, is a multidisciplinary body of work that focuses on the social organization of OSS and its implications. Included in this latter type of literature are studies of motivation/incentives (Hertel et al. 2003, Lakhani and von Hippel 2003), gift exchange (Bergquist and Ljungberg 2001, Zeitlyn 2003), innovation models (von Hippel and von Krogh 2003), coordination (Ljungberg 2000, Markus et al. 2000), and the economics of open source (Lerner and Tirole 2002). Yet, there is still much to be desired in obtaining a rich understanding of the social fabric of OSS projects. Acknowledging this, one of the items identified in von Hippel and von Krogh's (2003, p. 220) list of pointers to under-researched themes within OSS is the "interpretation of subtle matters relevant to organizational researchers [based on] a good contextual and behavioral understanding". The aim of this paper is accordingly to contribute to such an understanding and discuss implications for management that follow.

Thematically and methodologically, we promote a *process-oriented perspective*

(Langley 1999) on the social organization of OSS efforts emphasizing the importance of the small, seemingly mundane actions and gestures. The everyday life - not heroic moments or 'big' decisions - form the life-blood of OSS efforts and deserve closer scrutiny. For instance, there is the task of constantly delegating non-heroic, 'scrub' work. There is a significant amount of this work in any OSS effort which hardly is innovative in a traditional sense, thus rendering this type of work largely invisible, and giving rise to the issue of "who do the non-sexy work?" (Bonaccorsi and Rossi 2003, p. 1256). This implies that we bracket issues about whether a given decision, event or design represents an innovation, a redesign or simply a routine matter. In recognition that "working, learning, and innovating are interrelated and compatible and thus potentially complementary, not conflicting forces" (Brown and Duguid 1991, p.40), we embrace a stream of knowledge management and organizational learning literature that emphasise how innovation are part of daily 'non-canonical' practices, unexpected twists and turns, and the 'muddling through' of practical decision making and knowing (Orlikowski 2002).

Our perspective is also characterized by an emphasis on the *fragility* of OSS projects in the sense that we focus on the continuous maintenance or cultivation of the network of developers, users and software. Keeping the network of OSS developers together, we argue, is a performed achievement based on continuous social, ritual and technological means. This takes an ongoing effort, not a one-time shot, and includes recognizing the constitutive element of repetitions and various social and symbolic forms of reassurances. We aim at describing organizational, ritual and technological mechanisms that make up the everyday maintenance of OSS projects, the glue which keep them going by fending off centrifugal tendencies. To this end, in addition to literature from organization and management studies we draw on notions and insights from the two fields of science and technology studies (STS) (Latour 1996, Latour 1999) and social anthropology (Goffman 1967). Both of these latter

fields are also, of course, vast. We thus rely on a highly selective and targeted import outlined in more detail below.

Empirically, we base our argument on a case study of Gentoo Linux. Gentoo Linux is a non-commercial GNU/Linux distribution developed and maintained by the Gentoo community, currently comprising about 200 developers. Gentoo Linux is the fastest growing GNU/Linux distribution with an increased market share of 49 % since January 2004.

The remainder of this paper proceeds first with an outline of a process-oriented perspective on OSS, before addressing methodological issues. The Gentoo case is then presented, emphasizing three illustrative episodes. The succeeding section analyses three types (organizing, ritual, and technical) of mechanisms that are at play in keeping the Gentoo Linux project going. In the final section, the paper concludes by providing analytical, methodological and practical (for project management) implications.

A Process-oriented Perspective on OSS Efforts

There is a substantial and growing body of literature on organizational and managerial issues on OSS (Bergquist and Ljungberg 2001, Markus et al. 2000, von Hippel and von Krogh 2003). Guided by theoretical insights from STS focusing on technology and organizing as a *performed achievement* (Latour 1996, Latour 1999), as well as social anthropological studies of *interaction rituals* (Goffman 1967), we outline our analytical framework by positioning ourselves relative to dominant assumptions about key aspects of OSS. For reasons of clarity, this takes the form of dichotomous oppositions as shown in table 1 below.

Table 1: Key assumptions on the social organization of OSS projects.

Aspects of OSS	Dominant position	Our perspective

1. Process characteristics	Phase-oriented, life cycles, initiation rituals (rites de passage) (e.g. Jørgensen 2001, Mockus et al. 2002, von Krogh et al. 2003)	Continuous, small-step, interaction rituals, performed achievement
2. Involved actors	Top-notch programmers, elitist, non-profit, gift economy (e.g. Bergquist and Ljungberg 2001, Glass 2004, Lerner and Tirole 2002, Markus et al. 2000)	Decent programmers, commodification and gift exchange not dichotomous
3. Project characteristics	Robust projects, 'automatic' once started, private-collective innovation model, strong network externalities (e.g. Bonaccorsi and Rossi 2003, Dalle and Jullien 2002, Markus et al. 2000, von Hippel and von Krogh 2003)	Fragile projects, strong centrifugal forces, uncertain network externalities

Process Characteristics

There is a tendency in some OSS studies to emphasize relatively normative descriptions of the process in terms of key events, milestones or phases. For example, in his study of the FreeBSD project, Jørgensen (2001) focuses on the normative guidelines and how these can be captured in a "life cycle" of six different stages[1], rather than carving out the actual practices. Similarly, in their detailed report from the Apache and the Mozilla open source projects,

[1] code, review, pre-commit test, development release, parallel debugging and production release (Jørgensen, 2001: p. 328)

Mockus et al. (2002, p. 316) also argue that certain activities must be conducted before others, in order to overcome certain barriers in OSS efforts:

> "Apache began with a conscious attempt to solve the process issues first, before development even started, because it was clear from the very beginning that a geographical distributed set of volunteers, without any traditional organizational ties, would require a unique development process in order to make decisions".

How such complex process issues and the "unique development process" were achieved, however, is not elaborated. Furthermore, von Krogh et al. (2003) address the event of joining. They focus on the details of the novel practices within OSS projects. Drawing on an empirical study of the open source project Freenet, the authors develop an inductive theory concerning the different decision, steps, phases, and learning activities undertaken by newcomers when joining an OSS project. From a social anthropological point of view, this corresponds to a focus on the dynamics of ritual practice according to liminality and transitional phases – *rite de passage* (Turner 1969). Through transitional rituals individuals and groups become members and true participants of the society, achieve new statuses and may play new roles. In line with our ambition to illuminate ongoing, everyday practices of OSS, we draw on complementary conceptualizations of rituals. We rely on the notion that ritual actions are not only conducted in ritual contexts, and acknowledges that rituals are tied up with and partly constitutive of everyday social interaction, as outlined by Goffman's theories of interaction rituals (1967).

Involved Actors

A number of studies argue (or assume) that OSS developers are distinct from commercially based ones in terms of level of competence and motivation. Hence, Lerner and Tirole (2002, p. 206) conclude that "the open source process is quite élitist" as the top 15 developers

contribute 83-91% of the changes made in the source code. This perception of the competence of OSS developers seems to be linked to the methodological privileging of studying predominantly high-profile, success projects (Mockus et al. 2002). The huge majority of OSS projects, however, are neither high-profiled nor necessarily a success. At SourceForge.net[2] , the largest OSS development website, there are over 86.000 projects with over 900.000 registered developers. In these OSS projects, strong claims regarding level of competence of the developers need to be carefully argued rather than boldly assumed.

Raymond (2001) suggested early the metaphors of the bazaar and gift exchange to explain the difference from commercially based developers, a set of metaphors that has since triggered lively debates (Bergquist and Ljungberg 2001, von Hippel and von Krogh 2003, Zeitlyn 2003). The importance of gift giving is elaborated by Bergquist and Ljungberg (2001) who theorize this practice by drawing on Mauss' analysis of the social and cultural context of gift giving. In communities in which gifts are central, individuals status is not determined by what they own, but to a considerable degree by what they give away. Hence, the practice of gift giving is tightly related to issues of power and social control. Likewise, Bergquist and Ljungberg (2001) show how gift giving practices in open source projects are salient for getting ideas and prototypes into circulation among developers, and thus play an important role in structuring the very social relationships that make OSS projects possible. Bordering on the extreme, OSS has even been portrayed as resonating with "basic communist philosophies", where people are "working for no financial gain" (Glass 2004, p. 25). More soberly, however, empirical studies have developed more nuanced theories and explanations for what motivates individuals to participate on a volunteer basis (Franke and von Hippel 2003, Hertel et al. 2003, Lakhani and von Hippel, 2003, Markus et al. 2000, von Hippel and

[2] www.sourceforge.net accessed August 2004

von Krogh 2003). For instance, von Hippel and von Krogh (2003), suggest that OSS development activities embody what they call a "private-collective" activity. When giving away code for free, open source programmers not merely contribute to a public good in a purely altruistic sense, but simultaneously obtain private benefits in terms of learning, enjoyment, as well as working solutions to their 'private' technical problems.

Our perspective draws on a social anthropological stance where the dichotomous separation of gifts from commodities has been criticized and substituted by a more nuanced description where gifts and commodities are but the end points of a continuum, not constituting "universally some kind of unbridgeable chasm between gift and commodity exchange" (Parry and Bloch 1989, p. 10). Gifts and commodities are not principally different entities (thus warranting different analytical and methodological approaches), but vary only in degree and form (thus warranting the same approach). To assume up front that actors adhere to a 'gift' or 'commodity' economy should be avoided in favor of a study of the specific forms and degrees of exchange analyzed in terms of "combinations, associations, relationships and strategies for positioning" (Callon 1998, p. 12). In sum, then, we approach OSS developers without assuming up front that they differ dramatically from other developers.

Project Characteristics

One stream of the literature outlines and discusses the apparent rapid diffusion and strong momentum of OSS innovations despite a market dominated by proprietary standards and software (Bonaccorsi and Rossi 2003, Lerner and Tirole 2002). Observers have explained this by underscoring the immediate and long term benefits for the individuals involved (Lerner and Tirole 2002, Markus et al. 2000) combined with the positive network externalities inherently generated in (successful) OSS projects (Bonaccorsi and Rossi 2003, Dalle and

Jullien 2002, von Hippel and von Krogh 2003). In Markus et al. (2000, p. 15), it is argued that individuals have "multiple, reinforcing reasons for participating in open-source projects" in terms of both social and economical benefits. The expected social benefits related to gaining or enhancing one's reputation, whereas economical gains are expected to be generated through enhancement of products or adjacent services. Lerner and Tirole (2002) present a similar argument there they distinguish between "immediate benefits" and "delayed awards". Here the authors identify learning and enjoyable challenges as immediate benefits, and the possibility of future job offers, shares, and future access to venture capital as delayed awards. The source of the strong and positive network externalities are argued to stem from a critical mass of especially dedicated hackers combined with compatible standards and technologies (Bonaccorsi and Rossi 2003, Dalle and Jullien 2002). In a recent paper by von Hippel and von Krogh (2003), the individual and the collective-levels are drawn together in a common innovation model referred to as the "private-collective innovation model". Here the authors emphasize the positive externalities that are generated through a steady increase in private returns upon sharing of code. This leads to an increase in publicly shared source code (a public good) that reinforces the benefits for each individual participant, which again increases the amount of publicly shared source code, and so forth. A focus on how OSS gain momentum and exhibit strong, positive network externalities, at least implicitly, assumes a certain inertia of OSS projects: once underway, projects "automatically" diffuse. This is not so much an explicit assumption as expressed through a lack of emphasis on failed, halted or dying OSS projects. In this sense, we argue, it (implicitly) conveys a perception of OSS projects as robust.

Our perspective, drawing heavily on insights from science and technology studies (STS) (Latour 1996, Latour 1999), emphasizes the ongoing work – technical, social and symbolic – involved in making a project gain momentum. Projects, including OSS, need

constant attention and maintenance to avoid collapsing. Projects, on this account, are fundamentally fragile and fallible. As Latour (1996, p. 86) puts it, "for technology, every day is a working day".

Method

Our research process has proceeded through three phases: case selection, data gathering, and data analysis. Table 2 summarizes distinctive choices made in our research design. Again, this is for reasons of clarity presented as a dichotomy relative to a dominant approach.

Table 2: Summarizing key elements of our research design.

	Dominant approach	Our approach
Case selection	Success cases (Linux, Apache, Freenet), acclaimed technology.	Moderate success (Gentoo Linux), mundane technology.
Type of event	Decisive events, 'big' decisions, innovative design	Everyday practice, small gestures, routine development.
Source	Emails, memos (asynchronous)	Internet Relay Chat channels (synchronous)

First we selected case. Despite the overwhelming variety in type, size and history of OSS projects (illustrated by the 86.000 different projects hosted at SourceForge.net), the sampling of cases in the OSS literature does not seem to mirror this adequately. There is a preoccupation with a selective set of relative high-profiled cases such as Apache (Franke and von Hippel 2003, Mockus et al. 2002), Linux (Moody 2002), and Freenet (von Krogh et al.

2003). We have consciously selected a low-profiled case, the Gentoo Linux distribution, that so far has not been studied and which deals with mundane technology.

After case selection, we proceeded to data collection. One of the authors did a participant observation (Fetterman 1998) of the Gentoo OSS developers during a 4 month period. As data collection method, participant observation combines participation in the daily routines of the people studied with the researcher taking field notes and collecting non-observational data. The group of developers studied is not geographically co-located. Participating in their daily routines therefore meant seeking them where they interact with each other, meaning in this case 3 Internet Relay Chat (IRC) channels and two mailing lists. Focusing on the everyday practice of OSS development, participant observation consisted of reading the mailing lists daily in addition to spending between 4 to 12 hours every day following the developers on the IRC channels, observing and talking with them. During the 4 month period of observation complete transcripts of the IRC channels were automatically logged to disk. Transcripts were logged even when the researcher was not actively observing, providing 24 hours of transcripts every day. The transcripts were used to catch up on discussions in order to locate issue of interest while away from the IRC channels, and to follow up on these issues. After ended 4 months of observation, we had approximately 500,000 lines of IRC log transcripts.

A vital aspect of our research design, tightly linked with our research objectives, is the pivotal importance of IRC in our study. To study the everyday practice of OSS requires access to and the paying of due attention to the interactional, synchronous communication taking place. This is in line with von Hippel and von Krogh (2003, p. 220) when they note that "much user communication happens beyond public email. Thus Internet Relay Chat … can have a significant value for studies of motives, incentives, community development…".

Along with participating on the IRC channels, the two of the Gentoo mailing lists[3] were used while observing to locate issues of interest to follow up, much in the same way as the IRC transcripts were used. There was a significant overlap between issues discussed on the mailing lists and discussion on the IRC channels. Discussions on the mailing lists were often continued on the IRC channels, only to resurface later on the mailing list and visa versa. To get a better understanding of the developers' activities during the period of observation, ten informal e-mail interviews were performed with four key developers. These interviews were transcribed to facilitate analysis.

While doing participant observation field notes were taken daily based on observation and participation in the IRC channels along with the IRC transcripts, mailing list archives, and e-mail interviews. For contextual information, data was collected from three additional sources during the period of participant observation: the Gentoo web pages (http://www.gentoo.org), the bug tracking system (http://bugs.gentoo.org), and the revision control system (http://www.gentoo.org/cgi-bin/viewcvs.cgi/). These sources were used to supplement observations.

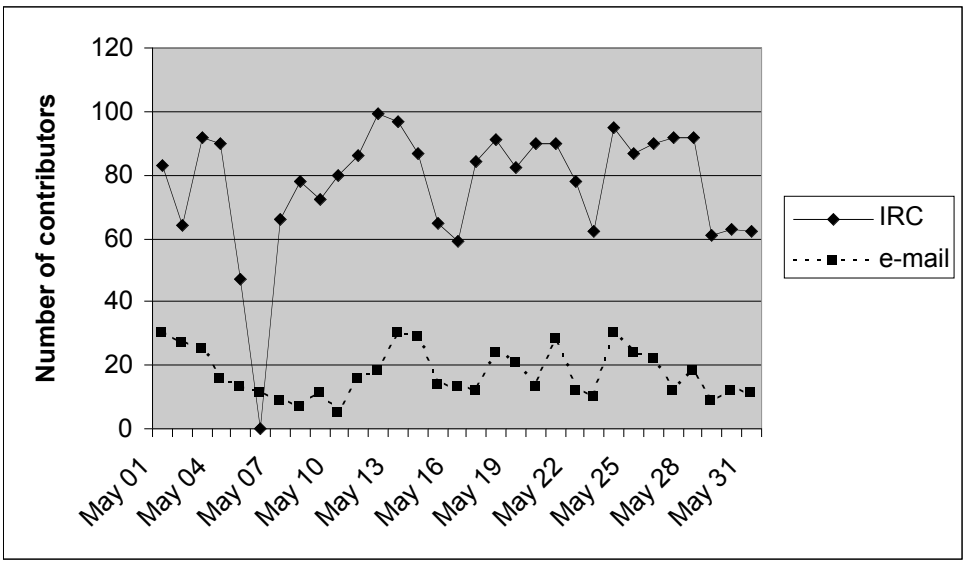
Based on the data sources above, we performed two forms of data analysis: a quantitative analysis of the IRC logs and mailing list, and qualitative analysis based on the field notes. First we performed the quantitative analysis. Based on data collected from the Web pages we generated descriptive statistics to show the amount of work connected with reporting and resolving bug reports. We also generated descriptive statistics to show the activity on IRC channels and mailing lists. Analysis of the IRC logs was performed with a simple data analysis tool[4] counting the number of contributors in an IRC log, the number of submissions per contributor, and the total number of submission within the log. We compared

[3] Accessed through http://gmane.org and nntp://news.gmane.org

[4] The tool is available from http://www.idi.ntnu.no/~thomasos/software/ila/.

the number of participants in the main developer mailing list and the main developer IRC channel as shown in figure 1 below. These statistics show that the developers use the mailing lists significantly less than the IRC channels for communication. That the developers' primary mode of everyday communication is IRC supports our initial research design.

Figure 1 Comparison of the number of contributors to IRC channel and mailing list[5]



Qualitative analysis was the performed. Through open coding (Strauss and Corbin 1999) we generated three categories of observations about how the effort was kept together: ritualistic, technological, and organizational. With basis in these categories we identified episodes (Langley 1999).

Case: The GNU/Linux Distribution Gentoo[6] Linux

Gentoo Linux is an open source GNU/Linux distribution. A GNU/Linux distribution is more

[5] The figure shows a drop of activity on the IRC channel May 6. This was caused by a network error preventing contact with the IRC server from 8:41 AM CET May 5 until 11:05 AM CET May 7. The drop is not an outlier, but rather the result of incomplete data.

[6] Calling the GNU/Linux distribution Gentoo alludes to the Linux penguin, the Linux mascot, as Gentoo is a species of small penguins living in large and noisy colonies on islands in the Antartic region.

than just the Linux kernel developed by Linus Torvalds. The Linux kernel is the interface between hardware and software, and needs to be integrated with 300-400 third-party libraries and applications to be a complete, usable operating system for office use. A software system that integrates the Linux kernel with third-party software is called a GNU/Linux distribution.

There are as of July 2004 over 200 active Linux distributions according to the linux.org web site. Of these, the market share of the commercial RedHat GNU/Linux distribution is 49,8 % and of Sun Cobalt Linux 20,3 %. Gentoo Linux has a 1.0 % market share[7]. Debian GNU/Linux, the most popular non-commercial GNU/Linux distribution, has a 15.9 % market share. However, according to the same analysis, Gentoo Linux is the fastest growing GNU/Linux distribution with an increased market share of 49 % since January 2004.

Gentoo Linux is a non-commercial GNU/Linux distribution developed and maintained by the Gentoo community, a loosely organized group of volunteers. Most GNU/Linux distributions provide more or less the same third-party software. However, Portage, Gentoo Linux' package manager, is specific to Gentoo Linux. The package manager is the software that automates installations and upgrades of third-party software. A lot of the development activities for Gentoo Linux consist in writing *ebuilds*, the installation scripts required by Portage to automate installation and upgrades.

Gentoo Linux started as a one-man open source project in 2000. By 2004 it is developed and maintained by a community of over 200 Gentoo developers. The developers are responsible for maintaining the software developed within the community (mostly Portage), and making sure that the third-party software supported integrates properly with Gentoo Linux. While many people contribute to Gentoo Linux, the title of Gentoo 'developer' is restricted. While there have been times when a general call for developers have

[7] According to the July 2004 Netcraft hosting provider switching analysis (http://news.netcraft.com/archives/2004/07/12/slight_linux_market_share_loss_for_red_hat.html).

been sent out, most are approached by a sponsor because of participation by Gentoo developers because of their participation. Becoming a Gentoo developer is therefore a transitory process of participation, finding a sponsor (or a sponsor finding you), answering a quiz to prove one's knowledge, gaining access privileges, and being officially announced as a developer with a given set of responsibilities. This is an informal process that is not described in any official documentation.

While still loosely organized, the Gentoo community adopted a more formal organization in mid-2003. This was "to solve chronic management, coordination and communication issues in the Gentoo project" (Robbins 2003). The new formal organization delegates responsibilities more clearly. Where maintenance responsibilities had previously been somewhat ad hoc, teams of developers, called herds, were now made responsible and accountable for the maintenance of specific parts of Gentoo Linux.

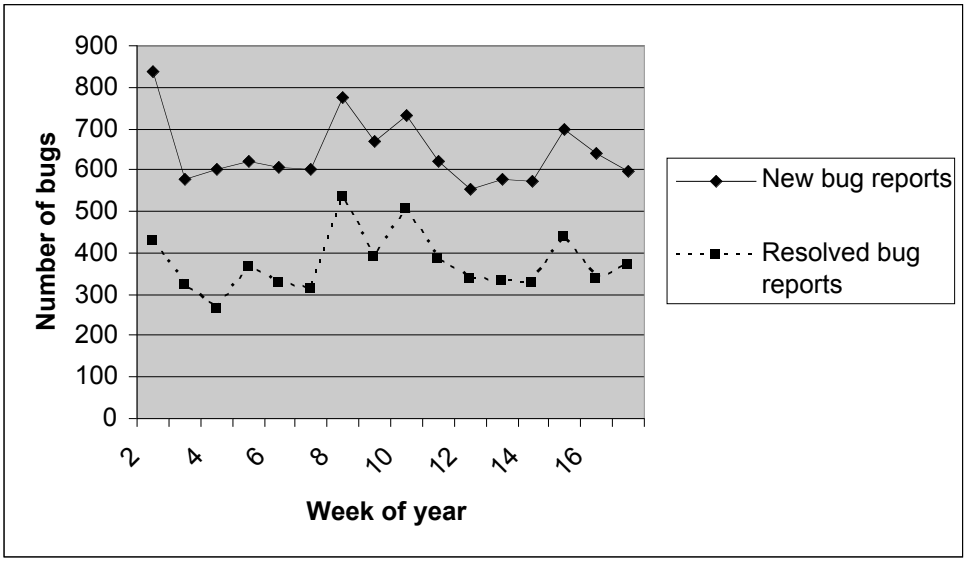
Episode 1: Handling bugs

The Gentoo developers have set up a bug tracking system to keep track of and make sure problems with Gentoo Linux are addressed and resolved. Problems are reported by filing a new bug report in the bug tracking system. Anyone may file a new bug report, update and read existing bug reports. The only requirement is that one registers with a username and password with the bug tracking system. When a new bug is filed in the bug tracking system, the bug report is given a unique identification number by the bug tracking system. It is then manually assigned to a developer or a herd to be resolved. The handling of bug reports is one of the Gentoo developers' main activities. Based on the weekly statistics[8], figure 2 indicates

[8] This is published in the Gentoo Linux Newsletter (http://www.gentoo.org/news/en/gwn/gwn.xml). After the week 17 bug statistics stopped being published for three weeks during late April to mid-May 2004, only to reappear again. This is why we have published statistics ending with week 17 2004.

the amount of work going into the effort of handling bug reports. On a weekly basis, the figure shows the number of new bug reports filed in the bug tracking system and the number of bug reports resolved. At most 837 new bug reports have been filed during a week. The average number of filed bug reports per week is 642. Keeping in mind that the Gentoo community consists of volunteers and that not all 200 developers are actively participating in the effort at any time, the Gentoo developers put a significant effort into handling bug reports.

Figure 2 Filed and resolved bug reports during the first 17 weeks of 2004



Resolving bugs is a process with extensive interaction between developers and the bug reporter. Brad, one of the Gentoo developers has an outburst on the Gentoo developers' IRC channel:

"I love some of these bugs nowadays. The bug is just spewing with information"

He then continues by doing what seems like a parody: "Ding dong the IM is dead! The IM is dead the IM is dead! Ding dong the wicked IM is dead!"[9] and continues with another parody: "Hi, I'd like to report a bug without doing a thought process as to what it might be and not

[9] IM is a instant messaging chat client

pasting my emerge --info ". Soon thereafter Sean, another Gentoo developer, joins in with his own complaints:

"The bug's marked critical. Yeah, right. Someone drop it to minor"

The source of Brad and Sean's complaints is a bug report recently filed in the bug tracking system. The bug tracking system provides a web form for filing new bug reports. The bug report in question is filed with this web form, and follows the conventions provided. A short description of the problem is provided, "IM is basically dead, you can't open an IM window without a crash", along with a more thorough description of the problem and the steps taken to reproduce the error. However, the bug report lacks three essential pieces of information. First of all, it does not say what version of the software this error occurs in. Second, it does not provide information about his system. Such information is retrieved using the 'emerge --info' command, which is the reference made in the second of Brad's parodies. Third, the bug report does not include the stack trace provided by the application when it crashes. The stack trace is the message provided by software to show where an error occurs.

There are two initial concerns when a new bug report is filed. The problem may be specific to the system where it occurs, or it may be a general problem. The first concern for the developers is therefore a question of reproducing the bug on another Gentoo Linux system with the same system configuration as that of the person filing the bug report. If the problem is reproducible elsewhere, the second issue is to determine if the problem is caused by the ebuild, the script to automate installation and upgrade of third-party software, or an error with the third-party software itself. If the problem is caused by the ebuild, the bug is resolved by the Gentoo developers. In the latter case, if it is not a problem with Gentoo Linux, it is what the Gentoo developers call an ‘upstream’ bug. The Gentoo developers do not participate in the open source projects that develop the third-party software integrated in Gentoo Linux. The Gentoo developers rarely fix bugs in third-party software. Their concern

is to correctly integrate software, not to make sure the software is working correctly in the first place. An upstream bug is therefore a bug that has to be resolved by the open source team developing the third-party software.

Without the stack trace and without knowing what version of the software the error occurs in, it is difficult to determine if this is an upstream error or a problem to be handled internally. The bug is "spewing with information", as Brad ironically puts it.

Shortly after this exchange, Sean logs on to the bug tracking system and updates the bug report. Not only has the person filing the bug report omitted important information, he has also assigned a critical severity rating to the bug report, which is the reference Sean makes in his comment "The bug is marked critical". Sean reduces the bug report's severity to normal in the bug tracking system, appending the following comment in the bug report:

"Severity changed to normal. This is not a critical bug"

A bug report's severity rating, however, can be misunderstood. In the web form for filing bug reports each severity rating is described. The critical severity rating is described as "The software crashes, hangs, or causes you to lose data". According to the bug report, the application does crash. The Gentoo developers, however, do not agree and assign it normal severity which, according to the web form, means "It's a bug that should be fixed".

Episode 2: Re-covering a bad fix

Of the Gentoo developers, only a small group works with developing and maintaining Portage, the Gentoo Linux package manager. These developers are organised in the Portage project. While still Gentoo developers, we differentiate the Portage project participants (dubbing them Portage developers) from the Gentoo developers at large. Portage is a keystone in Gentoo Linux as it automates the installation and upgrades of third-party software. It is therefore used by everyone running a Gentoo Linux system. This episode starts

with a new Portage release, release X. A few hours after release X is available, the following exchange takes place on the Portage developers' IRC channel:

Matz: Hmm. There is problem in Portage release X. John's fix and mine are in conflict.

John: Which fix?

Matz: Yeah, your fix is not bad.

A quick exchange of bug report numbers with explanations of how these bug reports were resolved and Matz concludes that:

Matz: Ok... I find out another solution about my fix. I've just changed it in my Portage. I'll test and commit it. John, can you tell devs not to use new portage in irc or mailing list...

John: Matz, you mean release X?

Matz: Yep

John: Matz, what bad things happen?

Matz: You can understand when you do 'emerge -p something'

John announces on the Gentoo developers' IRC channel that "no lovin' for portage release X just yet! two fixes for the same bug has created a new one." So that developers logging on the IRC channel will be informed, he then proceeds by adding the following statement to the IRC channel's topic:

"Portage release X is unfaithful - Stay away! For now"

Half an hour after the problem is discovered, Matz is back on the Portage developers' channel:

Matz: I've just fixed it in cvs. I tested it, but can anyone test it again? I have to do real

work now... :)

Now that the problem is resolved, attention is turned towards the damage the mistake may potentially have caused:

Graham: John, how long has release X been on rsync, and has it been masked in the meantime?

John: I'm about to mask it.

Matz: It was masked from first.

John: Ahh.. Of course :)

Graham: Well, must've been exposed less then 3-4 hours, since I went to bed around then and release X wasn't out yet :)

Matz: But Steve told dev to use masked portage

Graham: #gentoo-dev could use a topic change

John: already done it

Portage uses ‘rsync’, a program for copying files between computers, to download new ebuilds from a central ebuild repository. The longer "release X has been on rsync", the more people are likely to have downloaded the broken version of Portage from the central repository. However, downloading an ebuild is not the same as installing or upgrading the software. Portage release X is ‘masked’ to avoid that people install it. That software is masked means that while there exist an ebuild for the software, Portage will not install it. Masking is used when distributing experimental software to avoid accidental upgrade from a stable version to the experimental version. Software is unmasked by manually editing one of Portage's many configuration files. This is what Graham is point to when he says "Steve told dev to use masked Portage", where 'dev' is a typical shorthand for the Gentoo developers.

Since only the Gentoo developers are likely to have installed release X, John once again updates the Gentoo IRC developers' channel topic to include the statement "Portage release X wants to reinstall as much as possible - don't upgrade just yet". A few minutes later John once more changes the channel topic to include an explanation for recovering if release X has been installed.

Episode 3: Of bugs and men

The Gentoo managers' meeting was introduced with the mid-2003 reorganization. The managers' meeting is basically a forum for discussing and deciding on overarching decisions within Gentoo. In this episode the discussion is over a design issue within Gentoo Linux. Two Portage developers, Neal and Mark, argues the following design decision:

> "Our first order of the day is writing a **generally usable** abstraction layer over Portage's configuration files and database ... We have a slew of utilities lying about which hack, slash and mutilate the existing configuration files and misreads the database, so as to not be consistent with Portage itself. So we'll be looking at all the Gentoo specific tools in the Portage repository ... and throw away the seriously broken ones, and fix the rest to use the previously mentioned abstraction layer."

This design decision is proposed for a special class of third-party software, third-party software that is particular to Gentoo Linux in that it accesses Portage's database and configuration files. What makes this software special is that while it is tightly coupled with Portage's configuration file formats and database schema the software is upstream from Gentoo, i.e. it is developed and maintained by open source developers other than the Gentoo developers. Neal argues the problems as:

> "When we adopt a new package, are we supposed to be responsible for it, even if the upstream manager dies?"

This, however, is challenged by one of the other Gentoo developers:

> "These are Gentoo projects. So we are upstream in many cases. In case it is an internal project there should be a team within the tools top level project to maintain it."

Yes, Neal, agrees, the problem is not simply about upstream external software. It is also about software where the non-Portage Gentoo developers are responsible for maintaining the software. This, however, is not without problems as Neal explains:

> "There is a software utility that was written by Sergei when he was with us. He resigned, and I assumed maintenance. Then I had a leave of absence, and the baton was passed on. Now I have 20+ bug reports about this utility assigned to me. It's a mess, and nobody wants to touch it. Who is responsible to maintain it now? I guess I am ... so I replace it."

Then there is the issue of software that is maintained outside of Gentoo, as commented by another Gentoo developer:

> "Well when Tommy left he also said he won't maintain KP anymore as far as I know, but nobody seemed to care."

'KP' is a utility that accesses Portage's configuration files and database. KP, Neal continues, "was originally written by an external maintainer". "I guess it was popular", because it ended up in the ebuild repository. But the herd responsible for maintaining that part of Gentoo Linux, Neal continues, "being all of me and Mark, can't possibly retain maintainership of it". Neal's only claim to knowing how to maintain KP, is knowing the release manager of the graphical user interface library used to implement KP.

Analysis

As outlined in the theoretical section earlier, we emphasize how, as ongoing, performed

achievements, OSS projects are actively pushed forward on an everyday basis. This portrays OSS efforts as more fallible and fragile as they are threatened by disintegration in the absence of constant interactions. The navigating question for our analysis is the following: if the progress and success of OSS is not 'automatic', what are the mechanisms at play? Our analysis pursues three broad types of mechanisms: organizational, ritual, and technical.

Organizational mechanisms

The dominant position on OSS tend to portray, at least somewhat implicitly, OSS projects as ultimately self-strengthening and their organization as being relatively straightforward (Markus et al. 2000). As noted in the case section above, a recurring theme in Gentoo is the chronic shortage of manpower, and how working on Gentoo Linux consumes the Gentoo developers' time. A key driver here is the large and ever growing number of unresolved bug reports. The inability to process these as depicted in figure 2 above, accumulates a staggering work-load: in the 17 first weeks of 2004, over 5000 unresolved bugs accumulated, adding to the 1100 unresolved from 2003.

Thus, the developers in Gentoo are constantly dealing with the dilemma of balancing between doing exciting development work on the one hand, and the far less glamorous maintenance work on the other hand. Thus, constantly maintaining this balance, in the case of Gentoo the ability of the project to constantly re-organize and delegate 'non-sexy' work among developers has been crucial. The organization of how bugs are resolved is pivotal to keeping Gentoo going and is well worth closer scrutiny.

Users rarely know who to assign their bug reports to. Bug reports filed with no assignee are handled by to a group of Gentoo developers, the so-called *Bugwranglers*. The Bugwranglers' job is to distribute unassigned bug reports to the responsible person or herd for resolution. When assigned a bug report, the most imminent task of the assignee is to

determine whether the bug is specific to Gentoo Linux or an error in third party software (thus an upstream bug). The following dialogue shows some of the doubts raised when software is left un-maintained:

> **Mort**: There are still a lot of non-maintained GNU software in use. The standard GNU database library remains in version 1.8.3, and has not been updated in over two years.
>
> **Evan**: A lot of software is pretty much done and doesn't need any work.
>
> **Mort**: That doesn't mean there are no bugs in it.

Thus a crucial aspect in keeping Gentoo going is the everyday work of coordinating and delegating bugs. This kind of articulation work relates to what is often referred to as 'invisible work' in the field of science studies (Bowker and Star 1999). Although crucial for the daily health of the Gentoo project, this is 'invisible work' in the sense that neither the work roles (herd, Bugwrangler) nor their delegating work are particularly visible since it does not leave any traces in the source code itself. For understanding the OSS phenomena on an analytic level, it can thus be argued that this kind of invisible work needs to be taken into the account as part of the theorization. On the methodological side, this also underscores the importance of focusing on the real-time interactions typically happening on IRC channels, not so often visible on the mailing lists.

Ritual mechanisms

Previous research has suggested that newcomers to OSS projects go through different phases or 'joining scripts' before they are considered full-fledged members of the community (von Krogh et al. 2003). Following Goffman's (1967) interaction rituals of deference and demeanor – as complementary to transitional rituals – we discuss membership as complex interaction processes. By deference we understand the symbolic means of an activity by

which appreciation is conveyed to a person about the person. Deference is comprised of avoidance rituals – "those forms of deference which lead the actor to keep at a distance from the recipient…" (ibid. p. 62) – and presentational rituals. According to Goffman, presentational rituals, "encompasses acts through which the individual makes specific attestations to recipients concerning how he regards them and how he will treat them in the on-coming interaction" (ibid. p. 71). The symbolic significance of maintenance work emerges at times were developers start to doubt about the proper functioning of the software, or, even more highlighted, in situations of small crisis of breakdown and (near) accidents like illustrated above. In these types of occasions the fragile webs of interdependence in the project, especially manifested in the social organization of maintenance work, is socially emphasized and ceremonially celebrated.

While fixing the newly introduced bug in episode 2 is a way of fixing the software, notifying the rest of the developers about the problems is a way of ensuring their continued dedication to contributing with use and testing. It is a "specific attestation" to the fact that Matz and John's mistake has caused inconvenience to those developers using the masked software. The recipients of the attestation, those using the masked software, are told they are not lonely islands. The recipients are in the minds of Matz and John, and is important that the recipients continue testing the masked software. As such, keeping the other developers informed can be interpreted as a deference ritual.

The other main component of everyday ritual interaction is by Goffman identified as demeanor. Being a person of good demeanor is constituted exactly through the ritual interaction of performing demeanor. The individual creates himself as a respectable person through acting out the signs of deference, by salutations and giving compliments. If the person stops performing these signs, the decency and deference the person has created for himself will itself decline. Salutation as a sign produces to a large extent what the salutation

refers to, a polite person. The interaction rituals, as analysed by Goffman, needs constantly to be re-enacted to reproduce and maintain the social (and personal) phenomena they themselves are constituting.

In the Gentoo case developers perform demeanor by repeatedly informing their fellow developers and continuously sending acknowledgements. For example, while filing a bug report is an activity of providing the correct technical information and filling out a Web form, the bug report also plays a non-technical profoundly important symbolic role. In providing sufficient information, following the written and unwritten conventions for filing bug reports, the bug reporter presents himself as an insider, someone who knows the rules and is willing to play by them. Another illuminating example happened as one developer apologised for his failure to comply with the written and unwritten rules for filing bug reports and then provided the missing information. Through his demeanour, he presented himself as someone who wanted to partake in fixing the problem, not, as the Gentoo developers earlier identified him, as someone who were using others to solve his own problems and who could't be bothered to put an effort into helping resolve the troubles. Rather to the contrary, his deportment now explicitly and symbolically testifies his continuing commitment to the project.

Technical mechanisms

The technical maintenance of a technical artifact, like the Portage software, is sometimes portrayed as clearly delineated from development (Pressman 1997). This downplays the active role of the technology as a source of surprises and side-effects which subsequently need to be handled. Within science and technology studies, the presence of unplanned events is a well developed notion. For instance, Callon (1998, p. 38) points out how "any framing produces overflowing, and any procedure of disentanglement produces new attachments" and Latour (1999) underscores the "slight surprise" of any action.

The Gentoo case illustrates the influential role of such surprises and unintended side-effects. The fragility of the Gentoo project is demonstrated through the complexity of innumerable dependencies among the software systems many parts, the immense variation among installed GNU/Linux systems, and the multitude of different ways software is used increasingly produces side-effects (cf. episode 2 and 3).

Most OSS projects, including Gentoo Linux, are not self-contained products, but components of larger platforms. The presence of (unpleasant) surprises stemming from the Portage software thus also ties Portage developers together with other developers and projects. This implies that the Gentoo project is dependent upon other OSS projects and thus requires gifts from developers not initially concerned with Gentoo. While Bergquist and Ljungberg (2001, p. 308) make a strong argument arguing "that giving a gift brings forth a demand for returning a gift, either another object or, in a more symbolic fashion", the idea of open OSS as gifts, however, is too one-sided to explain the dynamics in the episodes above. While the gift economy argument may fit to explain the adoption of OSS by people outside projects, in our case projects upstream from Gentoo, we argue that the interaction rituals play an important role in the constant realigning of the development effort. It is a day-to-day maneuvering of actions and interactions with critical ceremonial components to keep the development effort running, beyond playing the role of putting others in debt by giving a gift. In fact, we argue that the ritual interaction plays the opposite role, of confirming that one is indebted to the recipient's efforts. In this sense, the gift economy argument and our use of ritual interaction complement each other.

Conclusion

OSS has, rightly so, attracted a considerable amount of attention. There is a tendency, though, in much of the OSS literature to exaggerate the uniqueness compared to other software

development (Curtis et al. 1988) and distributed project work (Orlikowski 2002). This is quite understandable in an early phase while struggling to make sense of a new phenomenon. We contribute to the growing recognition that this uniqueness should not be overstated, that the task now is to develop a more nuanced understanding that avoids overly dichotomous separation between OSS and other efforts. The implications that follow from our study are of three different types: analytical, methodological and practical.

First, analytically, OSS projects need to be conceptualized as related to, not independent of, other (software) development projects. For instance, the analytical separation where gift economy is applied to OSS while a commodity economy is relegated to other projects is highly unfortunate. Moreover, as shown by others (Curtis et al. 1988), also commercial software development typically involves coordination breakdowns, rituals, and unintended side-effects threatening the survival and stability of projects similar to OSS projects. Put differently, OSS initiatives are not necessarily stable than other software projects.

Methodologically, and in line with suggestions made by von Hippel and von Krogh (2003, p. 220), there is a need to focus more on the everyday interactions of OSS, including a closer scrutiny of interactive communication such as IRC. Our ability to follow crucial events and ongoing discussions in the Gentoo case was greatly enhanced by the real-time data provided on IRC channels. Qualitatively it gave us a more nuanced grasp of the everyday practice and rituals of OSS, and quantitatively it provided an overview of the activity within the project over time. In addition, the sampling of OSS cases should strive for a more representative picture than the one emerging from the present clustering around high-profiled, success cases.

On the practical side, our study has implications for the practice of project management. While conventional project management literature typically advocates a logic

of project control strongly emphasizing bureaucratic routines and follow-up of explicitly stated and agreed upon milestones, our study suggests that different forms of mechanisms are important for keeping large and high-risk projects together. Firstly, organizational mechanisms focusing on more loosely defined and situated divisions of labor combined with technological means for delegating and coordinating work seems to be necessary in distributed work contexts (Orlikowski 2002). Secondly, an important implication of our study is establishing that the various ritual mechanisms in terms of the small and often "invisible" interaction rituals expressed in the ceremonial aspects of maintenance of projects are important glue that prevents things from falling apart. The stability of open source projects, and possibly other knowledge intensive projects, is maintained through a stream of interaction rituals. Thus, from a management point of view, enabling a culture of communication that cultivates, includes and integrates all these types of minor forms of interaction rituals should be given due consideration – especially in the context of the dispersed work organization of contemporary "glocal" (Husted 2002) project based firms. Thirdly, in management of technology-infused projects one needs to take into account that the development and use of technologies also can be sources for uncertainties and surprises. On the other hand, as seen in the Gentoo case, in combination with organizational mechanisms, technological means for delegating and organizing work are also crucial for the survival of the project.

References

Bergquist, M., J. Ljungberg. 2001. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*. **11** 205-320.

Bonaccorsi, A., C. Rossi. 2003. Why Open Source can succeed. *Research Policy*. **32** 1243-1258.

Bowker, G., S.L. Star. 1999. *Sorting things out*. The MIT Press, Cambridge, MA.

Brown, J.S., P. Duguid. 1991. Organizational learning and communities-of-practice: toward a unified view if working, learning, and innovation. *Organization Science*. **2** 40-56.

Callon, M. 1998. Introduction: the embeddedness of economic markets. M. Callon, ed. *The laws of the market*, Blackwell Publishers, 1-57.

Curtis, B, H. Krasner, N. Iscoe. 1988 A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*. **31** 1268-1287.

Dalle, J., N. Jullien. 2002. *Open-Source vs. Proprietary Software*, Online paper at http://opensource.mit.edu/online_papers.php (Accessed August 2004).

Fetterman, D.M. 1998. *Ethnography, Second edition*. SAGE Publications, Thousand Oaks, CA.

Franke, N., E. von Hippel. 2003. Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software. *Research Policy*. **32** 1199-1215.

Glass, R.L. 2004. A Look at the Economics of Open Source. *Communications of the ACM*. **47** 25-27 .

Goffman, E. 1967. *Interaction ritual: essays on face-to-face behavior*. Pantheon Books, New York, NY.

Hertel, G., S. Niedner, S. Hermann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*. **32** 1159-1177.

Himanen, P. 2001. *The Hacker Ethic, and the Spirit of the Information Age*. Random House, New York, NY.

Husted, B.W. 2002. Cultural balkanization and hybridization in an era of globalization:

implications for international business research. M. Kotabe, P.S. Aulakh, eds. *Emerging Issues in International Business Research*, Edward Elgar Publishing, 81-95.

Jørgensen, N. 2001. Putting it all in a trunk: incremental software development in the FreeBSD open source project. *Information Systems Journal*. **11** 321-336.

Kesan, J.P., R.C. Shah. 2002. Shaping code. *Illinois Public Law and Legal Theory Research Paper Series*, Research Paper no 02-18.

Lakhani, K.R., E. von Hippel. 2003. How open source software works: "free" user-to-user assistance. *Research Policy*. **32** 923-943.

Langley, A. 1999. Strategies for Theorizing from Process Data. *Academy of Management Research*. **24** 691-710.

Latour, B. 1996. *Aramis – or the love of technology*. Harvard University Press, Cambridge, MA.

Latour, B. 1999. *Pandora's Hope*. Harvard University Press, Cambridge, MA.

Lerner, J., J. Tirole. 2002. Some Simple Economics of Open Source. *The Journal of Industrial Economics*. **2** 197-234

Levy, S. 1984 *Hackers: Heroes of the computer revolution*. Penguin Books, London, UK.

Ljungberg, J. 2000. Open source movements as a model for organizing. *European Journal of Information Systems*. **9** 208-216

Markus, M.L., B. Manville, C.E. Agres. 2000 What Makes Virtual Organization Work? *MIT Sloan Management Review*. **42** 13-26.

Mockus, A., R.T. Fielding, J.D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*. **11** 309-346.

Moody, G. 2002. *Rebel Code: Linux and the open source revolution*. Penguin Books, London, UK.

Orlikowski, W.J. 2002. Knowing in Practice: Enacting a Collective Capability in Distributed Organizing. *Organization Science*. **13** 249-273.

Parry, J., M. Bloch. 1989. Introduction: money and the morality of exchange. J. Parry, M. Bloch, eds. *Money and the morality of exchange*, Cambridge University Press, 1-32.

Pressman, R. 1997. *Software Engineering: A Practitioner's Approach*, Fourth Edition. McGraw-Hill Publishing, New York, NY.

Raymond, E.S. 2001. *The Cathedral and the Bazaar: Muslings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, Sebastapol, CA.

Robbins, D. 2003. Gentoo Linux Extension Proposal #4: Gentoo top-level management structure. Online at http://www.gentoo.org/doc/en/management-structure.xml (Accessed August 2004).

Strauss, A., J. Corbin. 1999. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, Thousand Oaks, CA.

Turner, V.W. 1969. *The ritual process: structure and anti-structure*. Routledge & Kegan Paul, London, UK.

von Hippel, E., G. von Krogh. 2003. Open source software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*. **14** 209-223.

von Krogh, G, S. Spaeth, K.R. Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*. **32** 1217-1241

Zeitlyn, D. 2003. Gift economies in the development of open source software: anthropological reflections. *Research Policy*. **32** 1287-1291