Selective Revealing in Open Innovation Processes: The Case of Embedded Linux

this version: March 2006

accepted for publication in Research Policy

Joachim Henkel

Schöller Chair in Technology and Innovation Management
Technische Universität München
Arcisstr. 21, D – 80333 Munich, Germany
phone: +49-89-28925741, fax: +49-89-28925742, email: henkel@wi.tum.de
CEPR, London

Abstract: *This paper provides a quantitative study (N = 268) of patterns of free revealing of firm-developed innovations within embedded Linux, a type of open source software (OSS). I find that firms, without being obliged to do so, contribute many of their own developments back to public embedded Linux code, eliciting and indeed receiving informal development support from other firms. That is, they perform a part of their product development open to the public—an unthinkable idea for traditionally-minded managers. Such openness obviously entails the challenge of protecting one's intellectual property. I find that firms address this issue by revealing selectively. They reveal, on average, about half of the code they have developed, while protecting the other half by various means. Revealing is strongly heterogeneous among firms. Multivariate analysis can partly explain this heterogeneity by firm characteristics and the firm's purpose behind revealing. An analysis of reasons for revealing and of the type of revealed code shows that different types of firms have different rationales for openness. Implications for management are that the conflict between downsides and benefits of openness appears manageable. Provided selective revealing is practiced deliberately, the opportunities of open development dominate.*

Key words: open source software, embedded Linux, free revealing, appropriation, IP protection

JEL classification: L86, M11, O31

1 Introduction

With some simplification, one can describe the traditional view of innovation as taking place entirely within one firm. In contrast to such closed innovation, open innovation processes are characterized as spanning firm boundaries (Chesbrough 2003). This may mean that technology is treated as a tradable good to be bought and sold on the market (Arora et al. 2001). However, openness in innovation processes can reach far beyond such market-mediated exchange. Under suitable circumstances, firms may make their technology available to the public in order to elicit development collaboration, but without any contractual guarantees of obtaining it.

Open innovation in this sense is the subject of the present paper. I explore the commercial development of open source software (OSS) for embedded systems such as machine controls or VCRs (e.g., VDC 2004). One of the benefits that firms can derive from using OSS is informal development collaboration (Feller and Fitzgerald 2002). Realizing this advantage requires that a firm reveals its code to the public—an obvious prerequisite for open innovation. Some OSS licenses work in the same direction by restricting means to keep code proprietary. However, these factors pushing for openness conflict with a firm's need to protect its intellectual property. So, how can open innovation be reconciled with intellectual property protection?

Using a quantitative empirical study of embedded Linux (N = 268), I explore how firms manage this conflict. First, I clarify that firms, despite the fact that Linux is OSS, indeed have a choice between openness and protection. I then analyze what share of their developments firms make public, what type of code they reveal, and what motivates them to do so. Using multivariate analysis I explore which firm characteristics determine revealing behavior. In particular, I investigate if and under what conditions openness leads to informal development collaboration, i.e., open innovation.

For several reasons, embedded Linux is ideally suited for studying the above questions. First, it is nearly exclusively developed by commercial firms, while hobbyists play only a minor role. Second, the fact that it comes under an OSS license makes firms consider revealing their own developments, which they would probably not even think of doing in the case of proprietary software. Still, they have considerable latitude in either sharing or protecting their code. As a result, openness is a conscious decision. Third, as one of the most

widely used operating systems in this field (Webb 2002, VDC 2004), Linux is of highest relevance for manufacturers of embedded devices. Such devices, in turn, account for the vast majority of all processors—around 6 billion in 2002 (Ganssle and Barr 2003). Hence, studying the innovation process of embedded Linux is not just instructive for understanding open innovation, but has implications for the large (and growing) embedded systems industry.

Central results are the following. Firms are aware of and routinely use various means of protecting their code. However, despite the possibility of protection they reveal on average about half of the code they develop for embedded Linux. The degree of openness turns out to be strongly heterogeneous among firms in my sample. Exploring this heterogeneity using multivariate analysis, I find that the share of its code a firm reveals is far from random. Instead, the analysis indicates rational cost/benefit considerations. In particular, the more important obtaining external development support is as a motive for free revealing, the more code the respective firm reveals. Furthermore, small firms ceteris paribus reveal significantly more, likely because, due to resource scarcity, they expect to benefit more from external development support.

Thus, open and collaborative innovation processes indeed take place. The private-collective model of innovation (von Hippel and von Krogh 2003) is found to work also in a commercial environment. However, firms practice "selective revealing" so as to minimize competitive losses—and are able to do so while abiding by the applicable OSS license. The patterns of free revealing I find are consistent with profit-maximizing behavior. It thus seems conceivable that OSS, even OSS under the GPL, becomes a standard part of industrial firms' innovative activity. The key is to understand what to reveal and what to protect—i.e., to repartition innovative activities into an open and a protected part in a manner consistent with private profits.

The remainder of the paper is organized as follows. In Section 2, background information is given on firms' benefits and downsides of developing OSS and on embedded Linux. Section 3 presents research design and data. In Section 4, analysis and results are presented. Section 5 concludes with a summary and a discussion.

2 Literature Review

The present paper links to four strands of literature: information trading, revealing of user innovations, collective invention, and commercial OSS development. In order of increasingly

close relation to this paper's subject, I will briefly review the relevant literature in the following, and will point out in what respect the present paper differs.

Information trading

Open information exchange between firms often occurs within a dyad of individuals. This phenomenon of "information trading" has been analyzed, among others, by von Hippel (1987) and Schrader (1991) and more recently by Dahl and Pedersen (2004). This literature finds that, despite the lack of formal contractual agreements, the information provider expects her counterpart to reciprocate when, in the future, she in turn requests information. A parallel to revealing OSS code is that in both cases the individual developer holds a gatekeeper position. The important difference, however, is that in a trading situation information is given to one particular recipient only. The provider thus knows if the conditions of acquaintance and mutual trust are fulfilled, which Bouty (2000) identified as preconditions for an interpersonal exchange of strategic resources in her study of R&D scientists. Furthermore, within a dyad a lack of reciprocation can clearly be attributed to one individual, and retaliation (in particular, ceasing to exchange information) can thus be targeted precisely.

Free revealing of user innovations

Studies of free revealing have to this point focused on free revealing by firms or individuals that expect to benefit from use rather than sale of their innovations. Free revealing by users has been found in fields as diverse as chemistry analyzers (von Hippel 1988), iron production (Allen 1983, see below), library information systems (Morrison et al. 2000), and sporting goods (Franke and Shah 2003). Those that freely reveal might be motivated to do so by expectations of benefit from development support by a user community (Franke and Shah 2003) or a manufacturer (Harhoff et al. 2003). Also building reputation among peers might be a powerful motivator, as Raymond (1999) argues for the case of (user-developed) OSS. In addition, users often lack the means to exploit their innovation by selling it, since the required change of functional role is typically difficult to accomplish (von Hippel 1988).

The development of OSS is indeed often performed by users (e.g., Franke and von Hippel 2003, von Hippel 2001). However, this need not be the case. As will be laid out below, firms may contribute to public OSS development for a variety of reasons. In particular, firms developing embedded Linux are typically *not* users of the software, but manufacturers (or suppliers to those) of goods containing embedded software.

Collective invention

Open innovation as I use the term in this paper is similar to the phenomenon of "collective invention", a particular instance of user innovation. The term has been coined by Allen (1983) in his study of iron production in 19th century England. During 1850-1875, two important attributes of iron furnaces were subject to steady improvement. Allen found that in many cases innovators publicly revealed data on their furnace design and performance in meetings of professional societies and in published material. He also documented a pattern in which innovators built on each other's advances. Nuvolari (2001a) describes another historical case of collective invention, namely, the development of the Cornish pumping engine. Further works on this topic are due to Cowan and Jonard (2003), Lamoreaux and Sokoloff (2000), McGaw (1987), and Russo (1985).

Various authors have drawn parallels between collective invention and OSS development (Henkel 2004a, Meyer 2003, Nuvolari 2001b, Osterloh and Rota 2004). Yet, there is an important difference. As Allen (1983, p. 2) points out: "Collective invention differs from R&D since the firms did not allocate resources to invention—the new technical knowledge was a by-product of normal business operation." This situation—one of innovation by *users*—corresponds to *user*-development of OSS (e.g., in the case of the Apache web server), but differs from that of embedded Linux. In the latter case, software development quite clearly *is* a part of R&D. Another difference is that, in the cases of collective invention mentioned above, all firms were following the same goal, namely, increasing efficiency of their furnaces and pumping engines, respectively. In contrast, it is precisely *heterogeneity* of needs which supports collaboration in the field of embedded Linux (Henkel 2004b). Thus, the phenomenon explored in this paper is related to, but clearly distinct from collective invention.

Costs and benefits of commercial OSS development

The existence of and nature of benefits commercial firms could derive from contributing to public OSS development have been explored by various authors (Behlendorf 1999, Hecker 1999, Raymond 1999, Feller and Fitzgerald 2002, Lerner and Tirole 2002, Wichmann 2002, Bonaccorsi and Rossi 2004, Dahlander and Magnusson 2005). Potential benefits that have been identified can roughly be grouped into four categories: setting a standard and enabling compatibility; increasing demand for complementary goods and services; benefiting from

external development support, in particular from the OSS community[1]; and signaling technical excellence or good OSS citizenship.

Against these possible benefits a number of potential downsides must be weighed. First, software that is freely available to anyone can no longer be sold—customers will at best be willing to pay for convenient packaging. Second, software revealed as OSS can be used also by competitors, which may imply a loss of competitive advantage. Third, the firm that originated the software might lose control over its further development, even when it acts as the official maintainer of the respective public OSS project.

As a result of balancing the above benefits and downsides, wide diffusion of the software and its source code may turn out to be desirable to the originator. In other situations, it may be preferable to keep the code proprietary—provided this choice exists. If the respective piece of software is based on existing OSS under the GPL, it must be put under that same license. In order to circumvent this requirement, the firm would have to abstain from using GPL'ed software in the first place.

How firms can realize the benefits while minimizing the downsides of public OSS development has been addressed by a number of authors. A recommendation for firms turning proprietary software into OSS is to choose "non-copyleft" OSS licenses (which would allow to reconvert future versions into proprietary software) or to use an OSS and a proprietary license in parallel (Behlendorf 1999, Hecker 1999, Raymond 1999).

The use of "standard" means of protection—legal mechanisms, secrecy, lead time, and complementary assets—in the case of OSS has been analyzed by Dahlander and Magnusson (2005) using case studies. They find that four out of five firms in their sample make sure to keep the copyright to the entire software program, thus maintaining a higher level of control over the software. Three of the firms close at least parts of the software, thus using secrecy and legal protection mechanisms. The remaining two firms keep their software open, relying instead on a committed developer community as a complementary asset. Note, however, that only the last of these solutions is viable in the case of derived work based on GPL'ed software. In a similar, but broader context West (2003) analyzes the success of semi-open platform strategies by Apple, IBM, and Sun.

---

[1] See Hertel et al. (2003) and Lakhani and Wolf (2005) for empirical studies of individuals' motives to contribute to public OSS projects.

A protection and appropriation means independent of the type of licensing is provided by complementary assets (Teece 1986). In the present context, this may be a brand name, as for distributors such as Red Hat and SuSE (e.g., Feller and Fitzgerald 2002); a developer community committed to the respective firm and its software (Dahlander and Magnusson 2005); proprietary software specific to the OSS in question; or hardware to which the respective OSS is tailored.

These results on commercial OSS development leave an important gap. First, several of the protection mechanisms discussed above are not available when a firm develops software based on existing OSS under the GPL. Given the wide diffusion of Linux and the dominance of the GPL among open source licenses (Lerner and Tirole 2005), this case obviously merits particular attention. Second, there is no study that analyzes the decision to reveal code or not on the level of individual code contributions—acknowledging the fact that firms might reveal selectively only some of their developments. Third and finally, there is no quantitative empirical study that relates a firm's revealing behavior to its characteristics, thus contributing to a deeper understanding of where open innovation works and where it does not. This paper addresses these issues.

3 Research Questions and Hypotheses

The central question of this paper is under what conditions open innovation processes are feasible—"open" in the sense that firms make their developments freely available to the public and receive informal development support from outside. The particular case I explore is that of embedded Linux. In this context, the following research questions will be addressed:

1. *What latitude do firms have with respect to revealing or protecting the code they develop for embedded Linux? What means of protection do they use, and how often?*
2. *What share of the code for embedded Linux that these firms develop is made public?*
3. *What type of code is typically revealed? What code is typically protected?*
4. *What are the reasons for firms to make their OSS code voluntarily public?*
5. *How is the degree of openness determined by firm characteristics?*

Since the context is the development of OSS under the General Public License (GPL, see Section 4.1) question 1 needs to be addressed first. Obviously, if a firm's revealing behavior was entirely determined by the GPL then the only decision of interest would be why

it uses OSS in the first place. Having established that firms do have a choice with regard to openness, the obvious next questions 2 and 3 relate to their actual revealing behavior. Questions 4 and 5 then deal with how this behavior can be understood and explained.

Question 5 will be addressed using regression analysis. To guide the choice of explanatory variables, hypotheses are derived from interviews (see 4.2) and the literature.

The share of code developed for embedded Linux that a firm reveals to the public is, ceteris paribus, larger …

H1: … the smaller the firm.

H2: … if there is a firm policy in place that encourages revealing.

H3: … if there is no firm policy restrictive to revealing in place.

H4: … if there are proprietary complementary assets available.

H5: … the longer the firm has been developing embedded Linux.

H6: … the more important, as a reason to reveal, considerations are regarding…

a) development support, b) marketing, c) reputation, d) the GPL.

Hypothesis H1 is derived from resource considerations. Active participation in the open source development process can provide firms with external development support as well as a low-cost marketing channel (Aldrich and Auster 1986, Gruber and Henkel 2005). Due to resource constraints, both should be relatively more important for small than for large firms. Hypotheses H2 and H3 are self-explanatory as far as the direction of the effect is concerned. However, it is not obvious that a significant effect can indeed be found. H4 derives from considerations by Teece (1986). Assuming that, for device and component manufacturers, their hardware constitutes a complementary asset to the embedded Linux code they develop I hypothesize that hardware manufacturers reveal, ceteris paribus, more than software firms. This effect should be more pronounced for component manufacturers since their code, typically drivers, shows a higher specificity to the respective hardware than in the case of device manufacturers.

H5 is posited because firms new to embedded Linux development typically will not yet have high-quality code to reveal. In addition, full adoption of the open source development process (and hence, revealing of code) requires a (presumably slow) change from the traditional proprietary attitude to an "open" one. This change should be slower, due to organizational inertia, the older a firm is at the time it starts developing embedded Linux (variable: AgeAtStart). On the other hand, a firm entering the embedded Linux arena with longer experience in a fully proprietary business model is more likely to have complementary

assets, and so might reveal *more*. Given these counteracting effects, I refrain from positing a hypothesis regarding the effect of AgeAtStart, but do include it as an explanatory variable. Finally, H6a through H6d relate to self-reported reasons for revealing. For example, the more important, *ceteris paribus*, development support (H6a) is for a firm the more it should reveal.

4 Research Design and Data

4.1 Object of Study: Embedded Linux

In recent years, more and more technical products have become "smart" in the sense of containing microprocessors and software. The number of processors built into such "embedded systems" dwarfs that of processors used in computers: of the 6.2 billion processors manufactured in 2002, more than 98% went into embedded systems (Ganssle and Barr 2003). Accordingly, software development constitutes an ever increasing part of overall new product development (NPD), even more so since embedded systems are highly heterogeneous and thus require more software adaptations than standard computers. In this situation, efficiency and effectiveness of software development become increasingly important (Rauscher and Smith 1995).

In this arena of embedded software, embedded Linux plays an increasingly important role. The term denotes versions of Linux used in embedded devices such as mobile phones, VCRs, and machine controls. It has experienced rapid development over the last years (Webb 2002) and has become one of the three most widely used operating systems for embedded devices (VDC 2004). Due to high heterogeneity of embedded devices, there is no standard version of embedded Linux. Correspondingly, "developing embedded Linux" refers to the development of modules or extensions that make Linux suitable for embedded systems. Examples are the *RTAI* real-time module, the toolkit *busybox*, the shrinked C library *uclibc*, and architecture-specific code for processors used in embedded devices.

Such code is to be regarded as "derived work" in the sense of the GPL, which governs the use of Linux. This implies that, by the time a device containing embedded Linux comes onto the market, the source code of the version of Linux it contains must be made available to all buyers. Obviously, this is a matter of concern for firms using GPL software. Still, considerable latitude exists with respect to sharing or protecting one's developments, as will be discussed in Section 5.1.

Further characteristics add to making embedded Linux a suitable object of study for my purpose. Nearly all developments come from firms—device manufacturers, component manufacturers, and dedicated software firms—while hobbyists play only a minor role. Hence, garnering support from hobby developers can be ruled out as a major motivator for firms to engage in this field of OSS. Furthermore, these firms benefit from contributing to embedded Linux not by increasing demand for complements nor by pursuing strategic interests as, e.g., IBM does by supporting Linux as a server operating system. Instead, embedded Linux constitutes, for device manufacturers, a part of their products. For that purpose, diffusion of the respective source code would not be required. Hence, if firms *still* voluntarily reveal their code then they must expect other benefits. As it will turn out, external development support from other firms is indeed the main motivator.

Embedded Linux thus offers an ideal opportunity to explore selective openness in innovation processes. This opening-up concerns the operating system, a product component which is of vital importance but typically not an important differentiator (even though this may be the case). The case of embedded Linux thus demonstrates a repartitioning of innovative activity into an open and a protected part. The decision that firms face is where to draw the line.

4.2 Data

Between November 2003 and March 2004 a web-based questionnaire on the development of embedded Linux was online. It had been developed based on a series of 30 interviews with industry-participants in the field of embedded Linux and yielded 268 valid responses.

The survey targeted developers, since they actually perform the act of revealing code and should thus be best informed about it. It was advertised on web portals and mailing lists dedicated to embedded Linux development. The basic population addressed by the survey thus consists of all embedded Linux developers who read the web portals or mailing lists advertising the survey.[2] Given that the questionnaire would have been hard to find and difficult to answer for people beyond this target group, responses from outside the basic

[2] Most participants (51.5%) learned about the survey on the portal LinuxDevices.com. Roughly 10% each came from the "PowerPC embedded" mailing list, the "RTAI" mailing list, and Handhelds.org. The "Busybox" mailing list accounts for 4.1%. 9% were informed by other sources (e.g., colleagues), and 6% did not disclose how they learned about the survey.

population seem unlikely. A precise response rate can not be given since data on the size of the basic population was not fully available. However, a nonresponse analysis comparing early to late respondents (Armstrong and Overton 1977) yields no indication of a non-response bias.

Asked to indicate what type of organization they work for, 22.4% of respondents described their employer as a "Software company specializing on embedded Linux", 42.5% as a "Device manufacturer", and 8.6% as a "Manufacturer of components like chips and boards". The remaining 26.5% of respondents ticked "I am working as a hobbyist" (15.3%) or "University or other non-profit research organization" (11.2%).[3]

Hence, only a minority of respondents conform to the cliché of the OSS developer as a hobbyist. Weighted by the number of hours per week spent on embedded Linux, this result becomes even more pronounced, with hobbyists contributing only about 7% of total hours in my sample. In addition, hobbyists are likely over-represented due to lower time pressure and higher emotional involvement (which is likely also true for university programmers). The notion that hobbyists play a minor role in embedded Linux development is thus supported.

Participants have considerable experience as programmers, on average 14.2 years. Average experience in developing OSS and embedded software, resp., is 4.9 and 7.1 years.

Those respondents working for commercial firms were asked some information on their employer. It turns out that software firms in the sample are comparatively young and small, with a median founding year of 1997 and 64% of respondents working for smaller firms (max. 50 employees). Device manufacturers have a median founding year of 1988, and 54% of respondents work for smaller firms. Component manufacturers, finally, are on average the oldest and largest firms, with median founding year of 1986 and only 26% of participants working for smaller firms. As to experience in developing embedded Linux, the distribution is strongly left-skewed, with very few firms (12.5%) having started between 1994 (the earliest date) and 1999. The mean starting year is 0.75 years earlier for software than for hardware firms ($p < 0.01$ in one-sided t-test).

---

[3] In order to avoid biased responses, participants were granted anonymity. In particular, they were not asked for the name of their employer. This means, of course, that some firms could appear more than once in the sample without being identified. Still, multiple occurrences seem to be rare. Most commercial developers (166 out of 197) had provided an email address. Of these, 83 could clearly be identified as company email addresses—and only three of them appear twice.

5 Results and Discussion

Following the research questions posited above, this section is divided into five parts. The first three subsections provide a descriptive analysis, addressing *means of protecting* OSS code (5.1), the *share* of code (5.2) and the *type* of code (5.3) that is actually revealed. The question *why* firms make their developments public is addressed in 5.4 using self-reported reasons. Finally, the central part of this study (5.5) provides a multivariate analysis of revealing behavior, allowing deeper insights into how firms balance openness and protection.

5.1 Ways to Protect Code Based on OSS

The GPL stipulates that the source code of derived work based on GPL'ed software must be made available to all receivers of the software. In the case of embedded Linux this means that when a device comes to market, any buyer is entitled to obtain the source code. This regulation, strict as it appears, nonetheless leaves room for protection.

First, contrary to a common misconception, derived work does not need to be made public. The GPL merely requires the seller of derived work—be it on a storage medium or embedded in a device—to make the source code available to its customers. If these are few, or even only one as in the case of commissioned OSS development, and if the customers themselves are not interested in revealing the code (and not obliged to by the GPL), then the software can effectively be kept secret. As Table 1 shows, 46.6% of respondents working for software firms stated their firm reveals code "sometimes" or more often only to its customers.

Second, even if a device is sold to the mass market, the developing firm can nonetheless delay and restrict diffusion by providing the source code on demand only, and without active support. Since typically more than a year passes between development of the code and market launch of the device, and since lead time is generally considered a rather effective means of protection (e.g., Sattler 2003), considerable advantage can be attained this way. Also this means is frequently used: 45.0% of all respondents working for commercial firms[4] stated that "sometimes" or more often their firm reveals code only when the device containing it comes onto the market, and only when buyers request it.

[4] Unless noted otherwise, all further percentages refer only to respondents working for commercial firms (N=197).

Third, the period of time between code development and market launch can be used to optimize the timing of active revealing. This means of protection—i.e., to actively reveal code, but only after a certain delay—was said to be used at least "sometimes" by 35.7% of respondents. Fourth and finally, it is accepted (if disputed) practice to make drivers available only as loadable binary modules, not as source code (Marti 2002). This turns out to be the most commonly used means, with 53.1% of respondents stating that their firm uses it at least sometimes; 16.1% even indicated "always". Further means of protection, which were not surveyed, must be applied before code is written as derived work of GPL'ed code. Often, re-designing the code architecture allows to shift to the (proprietary) application layer those functionalities that require protection (Henkel 2003).

Hence, despite the GPL various means to obtain a certain protection of one's code exist, and are routinely used.[5] Still, firms do reveal considerable amounts of code nonetheless, as will be reported in the following.

5.2 Revealing of Code—Extent and Change over Time

When quantifying the amount of code for embedded Linux that a firm reveals, a distinction is appropriate at the outset. For code that is so specific to a particular device that it is of no value for other firms, revealing (and also protection) is not an issue.[6] Hence, the corresponding survey question focuses on "*those embedded Linux developments by your firm that are potentially useful for others. That is, they are not too specific to your firm.*" "Revealing" was defined as follows: "*the code is actively made public, be it for downloading on a website, as a posting on a mailing list, or as a submission to the maintainer.*"

Table 2 provides descriptive statistics of the answers to the above question. Hobbyists and developers in universities exhibit "typical" open source behavior, revealing nearly all of

---

[5] Some parts of Linux are licensed not under the GPL but under the LGPL. When a firm links its own developments to such code, these can be kept proprietary more easily than in the case of GPL code. In my empirical study, I can not distinguish the two cases since information on the licensing schemes is not available. Still, due to the general dominance of the GPL—Lerner and Tirole (2005), in their empirical analysis of projects on SourceForge, count 18,133 projects under the GPL compared to 2,501 under the LGPL—it seems justified to focus the argument on code under the GPL.

[6] Morrison et al. (2000) report similar considerations in their study on user innovation on library computer systems.

the code in question (mean = 92%, median = 100%). In contrast, the mean across the three commercial categories is 49.3% (median = 50%). So, firms on average do make a considerable share of their developments public—49% is very much for an organization used to proprietary processes. However, they do not blindly follow the popular myth that derived work of GPL'ed code has to be made public.

A closer look at Table 2 shows that revealing behavior varies strongly between firms. For all three categories, the standard deviation of the share of revealed code is above 35%, the minimum at 1%, and the maximum at 100%. It will be the subject of Section 5.5 to explain some of this heterogeneity by firm characteristics.

One might conjecture that revealing in the field of embedded Linux is a left-over from the hype that surrounded OSS in 1999 and 2000, and that it decreases further over time. However, the survey yields exactly the opposite result. Only 10.3% of "commercial" respondents (total responses: 145) stated that their company reveals less now (i.e., at the time they did the survey, late 2003 / early 2004) than in 2000. 40.7% did not see a change, while nearly half of all respondents (49%) said their company reveals "somewhat more" or "much more" than in 2000. Hence, code sharing by commercial firms in the field of embedded Linux is anything but a dying-out left-over from a hype; quite the contrary, it seems to be on the way to becoming mainstream behavior.

5.3 Type of Code that is Revealed

Having established how code is protected and how much of it is revealed, the next question is what *type of code* firms typically reveal. The questionnaire offered five statements relating to the type of code, and participants were asked to indicate their agreement on a 5-point ordinal scale. Figure 1 shows the share of agreeing and disagreeing responses, separately for software and hardware firms.

The highest agreement, for both types of firms, received the statement that the revealed code is "generic"—with 63% agreement from hardware firms and even 85% from software firms. This is a very plausible finding, since revealing generic code should not harm the competitive position of the company (cf. Schrader 1991, Fauchart 2003).

Yet, this is not the whole story. Also the statements "is important for our competitive position" and "helps to differentiate our product from others" received a positive net agreement (i.e., share agreement minus share disagreement). While it is small in the case of

hardware manufacturers, it is considerable in the case of software firms (27.1% and 30%, resp.). A likely interpretation of this finding is that revealing for these firms is not at odds with, or may even support, differentiation and competitive advantage. This may be the case when revealing signals the firm's competences or when it creates demand for related services (for which the respective firm is best qualified) or related proprietary software.

The logic of complementary assets as a protection mechanism applies when the code is specific to some hardware or application software. Net agreement to the statement "is specific to our hardware" is relatively large (34.5%) from hardware firms; similarly, respondents working for software firms show a relatively high net agreement (28.6%) to the statement "is specific to our application software".

Two responses to the open question what types of developments the respondent's company would typically make public add a further perspective to the above statistical findings. The trade-off between costs and revenues linked to proprietary software is behind the first quote: "*[We reveal code] where the cost of support is more then the profit made on sales. We then give it out without support.*" The second quote addresses the protection of competitive advantage: "*Anything that will not give our competitors the possibility to copy our products one to one in a short time.*" Against this (and other) downside, firms weigh a number of potential benefits, which are dealt with in the following.

5.4 Reasons to Reveal

Based on the interviews I conducted as well as a literature analysis twelve potential reasons why firms would reveal their code were identified. Participants were offered these reasons and were asked to indicate their agreement to the statements "*My company reveals code because [reason x]*" on a 5-point ordinal scale. Results are shown in Figure 2, for hardware manufacturers only. This differentiation yields a clearer picture because hardware and software firms differ considerably with respect to some of these reasons. The focus is laid on hardware manufacturers since they are more numerous (137 vs. 60) in the sample. I will address the corresponding results for software firms at the end of this section.

The highest agreement received the statement "My company reveals code because the GPL requires it." This was to be expected—despite ways to circumvent it, the GPL obviously

does have a strong effect on revealing behavior.[7] The four reasons following on ranks 2 to 5 received only slightly less net agreement. Three of them—bugfixes by others, further development by others, and reduced maintenance effort—are directly related to informal outside development support, that is, to the technical benefits of the open source process. The remaining one, on rank 2 ("to appear as a good OSS player"), is indirectly related to that same aspect, since receiving informal outside support requires to be regarded as a good open source player (Osterloh et al. 2001, Franck and Jungwirth 2003). Hence, firms do perceive technical benefits from the open source development process in embedded Linux, and are willing to share their code with others in order to realize these benefits.

For the reasons on ranks 6 and higher, the level of agreement drops significantly compared to rank 5. On rank 6, still receiving twice as much agreement as disagreement, a reason related to marketing appears: "revealing good code improves our company's technical reputation." Further details can be obtained from Figure 2.

An analogous analysis was carried out for software firms. The main difference is that reasons related to marketing ("revealing good code improves our company's technical reputation" and "visibility on the mailing list is good marketing") rank much higher (ranks 3 and 6, in contrast to ranks 6 and 10 for hardware manufacturers). This can be explained by the fact that these software firms act as suppliers to hardware manufacturers, performing commissioned development and/or selling tools or specific distributions of embedded Linux.

In order to check consistency of responses and to construct meaningful indices to be used in the subsequent analysis (see 5.5), an exploratory factor analysis is carried out. With four components, it explains 64.6% of total variance and yields good quality measures (KMO: 0.70, $p < 0.001$). The resulting components can be interpreted as "development support" (ranks 3, 4, 5, 7, 9 in Figure 2), "reputation" (ranks 2, 6), "marketing" (ranks 10, 11, 12), and "GPL" (rank 1).[8]

---

[7] Given the survey's definition of "revealing" (see 5.2), the GPL does not require revealing at all, strictly speaking. Still, the requirement of the GPL to make the code *eventually* available may constitute a reason to do so *proactively*. In any case, this reason being on rank 1 does not affect the findings on the importance of the other reasons.

[8] The factor analysis uses principal component analysis and Varimax rotation. Cronbach's α for the components development support, reputation, and marketing obtains, respectively, as 0.80, 0.64, and 0.60. Compatibility as a reason to reveal (rank 8) can not clearly be attributed to any one factor since all factor loadings are below 0.5.

5.5 Multivariate Analysis of Revealing Behavior

The descriptive analysis of the amount of revealed code showed high heterogeneity between firms (Table 2). In the following, I present a multivariate analysis that aims at explaining this heterogeneity by firm characteristics. Due to missing values mainly in the dependent variable, the number of observations is reduced from 197 (number of commercial respondents) to 119.[9] Table 3 shows descriptive statistics of the explanatory variables, Table 4 the correlation matrix.[10] As to the type of firm, device manufacturers are taken as the reference group since they exhibit the lowest average share of revealed code (see Table 2).

Since the share of revealed code is restricted to the interval [0%,100%], a Tobit model is the natural choice.[11] In order to check robustness of results with respect to the model specification, I additionally use an Ordered Probit model. In this model, the dependent variable lies in the set {1, …, 5} where "1" indicates that the share lies in the first quintile (0% – 20%), "2" codes "21% – 40%" etc. This model allows for a non-linear dependence of the share of revealed code on the explanatory variables inside the interval [0%,100%].

Table 5 shows the regression outcome. Specifications (5) through (8) contain, in contrast to (1) – (4), indices "Reason_XY" (based on the factor analysis performed in Section 5.4) which indicate how important the respective group of reasons is for revealing. Specifications (1), (2), (5), and (6) employ a Tobit model, while (3), (4), (7), and (8) are based on an Ordered Probit estimation. Finally, specifications with even numbers are obtained by successive elimination of insignificant variables. In all cases, the hypothesis that the

---

Factor analysis for software and hardware firms separately yields slightly different components, which mirrors the specificities of the two groups. Only the pooled analysis is reported since its output will be used in 5.5.

[9] Missing values in the variables related to reasons for revealing (see Figure 2) would reduce this number further to 105. In order to avoid this loss, missing values were estimated using the *impute* command in Stata 9 for those observations in which no more than four out of twelve values were missing.

[10] One might propose to use also the type of code that a firm typically reveals (see 5.3) as an explanatory variable. However, as a second dimension of revealing behavior (in addition to the extent of revealing) it would rather have to be treated as a *dependent* variable in another regression, which is beyond the scope of this paper.

[11] In contrast to an OLS regression, a Tobit model accounts for the censoring of the dependent variable. In the present case this means that the share of revealed code can not be less than 0%, nor larger than 100%.

eliminated variables are also jointly equal to zero can not be rejected.[12] Comparing the columns of Table 5 one finds that signs and significances are, with a few exceptions regarding significance levels, consistent across the various specifications.[13]

Firm size: The regression results confirm Hypothesis H1. The coefficient for the dummy variable "SizeLarge" (indicating that the firm has more than 200 employees) is significantly negative in all specifications. This finding is consistent with the high level of agreement (90.0% of N=185) that the statement "Open source software allows small enterprises to afford innovation" received. The dummy variable "SizeMedium" coding medium sized firms (11 – 200 employees) is insignificant; obviously, this group is not different enough from the reference group (1 – 10 employees).

Firm policies: Two dummy variables capture the effects firm policies might have on revealing: "PolicyEncouraging" equals 1 if the respondent ticked the statement "*My company encourages me to contribute source code that is not critical for competition*" and "PolicyRestrictive" equals 1 if the person agreed to "*My company is very restrictive in revealing code*". While "PolicyEncourages" does not exhibit the expected positive effect (thus not confirming H2), "PolRestrictive" carries a significantly negative coefficient in specifications (1) through (4). Its significance disappears when, in specifications (5) through (8), reasons to reveal are introduced as explanatory variables. This is a plausible finding since, as the correlation matrix (Table 4) confirms, policies towards revealing should result from the importance attached to various reasons to reveal. H3 is thus partly supported.

Complementary assets: Since the descriptive analysis (Table 2) revealed that device manufacturers make, on average, less code public than the other two commercial groups, they were chosen as the reference group. This already suggests what the regression clearly shows, namely, that H4 can not be fully confirmed. Despite the hypothesized protection by a complementary asset (the device), device manufacturers reveal, in all specifications,

---

[12] A Likelihood Ratio test for the Tobit regressions yields $F(2,111) = 0.12$ ($p = 0.889$) for specifications (1)/(2) and $F(5,107) = 0.45$ ($p = 0.812$) for specifications (5)/(6). A Wald test for the Ordered Probit regressions yields $\chi^2(2) = 0.12$ ($p = 0.944$) for specifications (3)/(4) and $\chi^2(5) = 3.27$ ($p = 0.658$) for specifications (7)/(8).

[13] The pseudo R^2 value seems rather low for all specifications. However, a pseudo R^2 can not be interpreted as an R^2 in an OLS regression. To give an idea of the size of the explained variance, a standard OLS regression of specification 5 (leading to roughly the same coefficients and significances as the Tobit model) yields an R^2 of 0.27. This still leaves much variance unexplained—as expected—but is a much more reasonable value than 0.02.

significantly *less* than software firms. This may be due to the fact that hardware manufacturers are more entrenched in a proprietary culture than software firms (especially those that specialize on OSS), and/or that their hardware does in fact *not* constitute a sufficiently protective complementary asset.[14] In contrast, comparing software firms to component manufacturers does lend some support to H4. Across all specifications, the estimated coefficient of TypeCompMan is roughly 1.5 times as large as that of TypeSWFirm. However, the hypothesis of equality between the two coefficients cannot be rejected (for all specifications). H4 *is* confirmed, however, insofar as component manufacturers reveal significantly more than device manufacturers (the reference group). In addition to the higher specificity of their complementary assets (as conjectured), also demand from component buyers for driver source code may serve to explain this finding.

Familiarity with embedded Linux: The variable CompYearsEL, capturing a firm's experience in developing embedded Linux, carries a significant (5% level) and positive coefficient in all specifications. H5 is thus confirmed.

Company age when it started embedded Linux: The variable AgeAtStart is positive and significant (on the 5% resp. 10% level) in all specifications. Hence, of the two counteracting effects discussed above the argument related to accumulation of complementary assets seems to dominate.

Reasons to reveal: Of the four variables relating to reasons for revealing, those related to marketing and to the GPL do not show a significant effect. H6c and H6d can thus not be confirmed. A plausible interpretation is that firms for which marketing is more important as a reason to reveal may contribute code in a more visible way, but not in higher quantity. Similarly, firms that consider the GPL's stipulations as important reasons for revealing may in fact be quite reluctant to make their code public, and only do so when indeed they feel forced to. In contrast, the variables Reason_Development and Reason_Reputation carry positive coefficients in all specifications (with one exception), significant on the 5% resp. 10% level.

---

[14] In fact, a device manufacturer might combine commodity hardware with specifically developed software, based on OSS, to create a highly differentiated good. In one of my interviews, the interviewee described the case of a device manufacturer who had developed a board for bundling 48 UMTS channels. All elements of the board were commodities. In contrast, the driver software contained important intellectual property, in particular proprietary protocols. Had it been publicly revealed as OSS, competitors could easily have copied the device. In such a case, the hardware, as a complementary asset, obviously affords little protection to the software.

The effect sizes of both are large, and nearly identical. For illustration, an increase of 1 in the value of Reason_Development (e.g., a change from "agree somewhat" to "agree strongly" in the answers to the reasons that make up this index, see 5.4) is predicted to result in an increase of the share of revealed code of 8.8% (specification 6). H6a and H6b are thus confirmed.

To summarize, the regression analysis provides a lucent picture of firms' revealing behavior. In particular, confirmed Hypotheses H1 and H6a underline the importance of external development support as a motivator to reveal one's own code.

6 Conclusion

There is a long-standing debate on how profits from innovations can best be appropriated.[15] This discussion usually makes an implicit a priori: It presupposes that exclusivity is desirable for the innovator. It thus focuses on the *protection of innovations*, while what actually matters is *appropriation of profits* from innovation. The two often go along with each other—but there are important exceptions. As the rise of OSS has impressively shown, freely revealing one's developments may be a sensible thing to do, in particular when community-based development is viable (von Hippel 2001). This is favored when, as for embedded Linux, needs are strongly heterogeneous (cf. Bessen 2001, Franke and von Hippel 2003, Henkel 2004b) and the underlying technology is highly modular (Baldwin and Clark 2003, MacCormack et al. 2004).

On the other hand, developing OSS is often considered to *imply* free revealing. As I point out in this paper, also this view is not correct: Commercial OSS development, even if based on GPL'ed software, perfectly well accommodates a combination of free revealing and various means of protecting one's code. Firms thus have the chance to practice *selective revealing*.

A case where such combination of revealing and protection is common has been analyzed in this paper, namely, embedded Linux. Firms in the sample routinely use various means of protection that are consistent with the GPL. Among them are restricted or delayed revealing and the use of "binary-only" drivers. As a result firms reveal, on average, about half of the code they have developed for embedded Linux, while protecting the other half.

---

[15] See, e.g., Arundel (2001), Cohen et al. (2000), Cohen et al. (2002), Gallini and Scotchmer (2002), Harabi (1995), Horstmann et al. (1985), Levin et al. (1987), and Sattler (2003).

Between firms, revealing behavior is strongly heterogeneous, which can partly be explained by firm characteristics. Among other things I find that the amount of revealed code *ceteris paribus* is larger for smaller firms, which likely benefit more from external development support. It is also higher for component manufacturers, likely because these firms enjoy a good protection by proprietary complementary assets (the components they sell). Finally, experience with OSS matters: the longer a firm has been using embedded Linux and the higher hence its familiarity with the OSS development process, the more it reveals.

Among the reasons to reveal, informal outside development support in the form of bugfixes, code improvement, and code maintenance figures prominently, and turns out to be a significant driver of revealing. Hence, firms indeed seem to derive technical benefits from the open source development process, increasing efficiency and effectiveness of embedded software development. Thus, even in this mainly commercial environment, the private-collective model of innovation described by von Hippel and von Krogh (2003) is found to work.

A point that merits discussion and further research is the role played by the individual developer. One might conjecture that revealing by firms is in fact driven by OSS enthusiasm of the programmer—possibly even against his employer's interests. However, neither the survey nor my interviews support this view. Employed programmers *are* somewhat enthusiastic about OSS, but significantly less so than hobbyists and university programmers. While a detailed discussion is beyond the scope of this paper, there are indications that programmers do often act as "champions of revealing", but not as uncontrollable OSS enthusiasts disregarding their firm's interest.

Open innovation processes as observed in the field of embedded Linux carry a considerable potential for efficiency gains (Foray 2004, Henkel and von Hippel 2005), in a similar way as open science supports the accumulation of knowledge (David 2005). They allow to reduce duplication of effort and to avoid transaction costs of commercial licensing. Embedded Linux being OSS and being licensed under the GPL certainly helped to spawn the observed open innovation process. However, the fact that firms are far more open than they are obliged to by the license shows that *voluntary* revealing takes place. Hence, open innovation as observed here should just as well be feasible for software other than OSS, and for goods other than software. The main obstacle seems to be that most firms are entrenched in a proprietary attitude and unfamiliar with openness. As one respondent phrased it, "*[revealing code] is new territory for this company. The spirit is willing, but the legal staff is*

weak." This paper suggests that this attitude needs rethinking. Firms can benefit from open innovation by striking the right balance between sharing and protection.

References

Aldrich, H., E. Auster, 1986. Even dwarfs started small: Liabilities of age and size and their strategic implications. In: L. Cummings, B. Staw (eds.) Research in Organizational Behaviour. San Francisco, CA: JAI Press. 165-189.

Allen, R. C., 1983. Collective Invention. Journal of Economic Behaviour and Organization 4, 1-24.

Armstrong, J. S., T. S. Overton, 1977. Estimating nonresponse bias in mail surveys. Journal of Marketing Research 14, 396-402.

Arora, A., A. Fosfuri, A. Gambardella, 2001. Markets for technology: The economics of innovation and corporate strategy. Cambridge, MA: MIT Press.

Arundel, A., 2001. The relative effectiveness of patents and secrecy for appropriation. Research Policy 30(4), 611–624.

Baldwin, C. Y., K. B. Clark, 2003. The architecture of cooperation: how code architecture mitigates free riding in the open source development model. Working Paper, Harvard Business School.

Behlendorf, B., 1999. Open source as a business strategy. C. Dibona, S. Ockman, M. Stone, eds. Open-sources: Voices from the open source revolution. Sebastopol, CA: O'Reilly, 149-170.

Bessen, J., 2001. Open source software: free provision of complex public goods. Working Paper 5/01. http://www.researchoninnovation.org/opensrc.pdf.

Bonaccorsi, A., C. Rossi, 2004. Comparing motivations of individual programmers and firms to take part in the open source movement. From community to business. Working paper Sant'Anna School of Advanced Studies Pisa.

Bouty, I., 2000. Interpersonal and interaction influences on informal resource exchanges between R&D researchers across organizational boundaries. Academy of Management Journal 43(1), 50-65.

Chesbrough, H., 2003. Open Innovation: The New Imperative for Creating and Profiting from Technology. Boston, MA: Harvard Business School Press.

Cohen, W. M., A. Goto, A. Nagata, R. R. Nelson, J. P. Walsh, 2002. R&D spillovers, patents and the incentives to innovate in Japan and the U.S. Research Policy 31(8-9), 1349–1367.

Cohen, W. M., R. R. Nelson, J. P. Walsh, 2000. Protecting their intellectual assets: appropriability conditions and why U.S. manufacturing firms patent (or not). NBER Working Paper No. w7552.

Cowan, R., N. Jonard, 2003. The dynamics of collective invention. Journal of Economic Behavior and Organization 52(4), 513-532.

Dahl, M. S., C. O. R. Pedersen, 2004. Knowledge flows through informal contacts in industrial clusters: myth or reality? Research Policy 33(10), 1673-1686.

Dahlander, L., M. G. Magnusson, 2005. Relationships between open source software companies and communities: Observations from Nordic firms. Research Policy 34(4), 481-493.

David, P. A. 2005. The economic logic of "open science" and the balance between private property rights and the public domain in scientific data and information: A primer. SIEPR Discussion Paper No. 02-30.

Fauchart, E., 2003. On knowledge sharing patterns among rival firms: The case of knowledge on safety. Working Paper. http://userinnovation.mit.edu/papers/safety3.pdf.

Feller, J., B. Fitzgerald, 2002. Understanding Open Source software development. Boston, MA: Addison Wesley.

Foray, D., 2004. Economics of Knowledge. Cambridge, MA: MIT Press.

Franck, E., C. Jungwirth, 2003. Reconciling investors and donators—The governance structure of open source. Journal of Management and Governance 7, 401-421.

Franke, N., S. Shah, 2003. How communities support innovative activities: An exploration of assistance and sharing among end-users. Research Policy 32(1), 157-178.

Franke, N., E. von Hippel, 2003. Satisfying heterogeneous user needs via innovation toolkits: The case of Apache security software. Research Policy 32(7), 1199-1215.

Gallini, N. T., S. Scotchmer, 2002. Intellectual Property: When is it the best incentive system?, in: A. Jaffe, J. Lerner, S. Stern (eds.), Innovation policy and the economy, vol. 2 , Cambridge, MA: MIT Press.

Ganssle, J., M. Barr, 2003. Embedded Systems Dictionary. Lawrence, KS: CMP Books.

Gruber, M., J. Henkel, 2005. New ventures based on open innovation—an empirical analysis of start-up firms in embedded Linux. International Journal of Technology Management, accepted for publication.

Harabi, N., 1995. Appropriability of technical innovations – an empirical analysis. Research Policy 24(6), 981–992.

Harhoff, D., J. Henkel, E. von Hippel, 2003. Profiting from voluntary information spillovers: How users benefit by freely revealing their innovations. Research Policy 32, 1753–1769.

Hecker, F., 1999. Setting up shop: The business of open-source software. IEEE Software 16(1), 45-51.

Henkel, J., 2003. Open-Source-Aktivitäten von Unternehmen—Innovation in kollektiven Prozessen. Unpublished Habilitation thesis, University of Munich.

Henkel, J., 2004a. Open source software from commercial firms – tools, complements, and collective invention. Zeitschrift für Betriebswirtschaft, Supplement 4, 1-23.

Henkel, J., 2004b. The jukebox mode of innovation – a model of commercial open source development. CEPR Discussion Paper 4507.

Henkel, J., E. von Hippel, 2005. Welfare implications of user innovation. Journal of Technology Transfer 30(1/2), 73-87.

Hertel, G., S. Niedner, S. Herrmann, 2003. Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux kernel. Research Policy 32(7), 1159-1177.

Horstmann, I., G. M. MacDonald, A. Slivinski, 1985. Patents as information transfer mechanisms: To patent or (maybe) not to patent. Journal of Political Economy 93(5), 837-858.

Lakhani, K. R., R. G. Wolf, 2005. Why hackers do what they do: Understanding motivation and effort in free/open source software projects, in: J. Feller, B. Fitzgerald, S. Hissam, and K. R. Lakhani (eds.), Perspectives on Free and Open Source Software. Cambridge, MA: MIT Press.

Lamoreaux, N. R., K. L. Sokoloff, 2000. The geography of invention in the American glass industry. Journal of Economic History 60(3), 700–729.

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

Lerner, J., J. Tirole, 2005. The scope of open source licensing. Journal of Law, Economics and Organziation 21, 20-56.

Levin, R. C., A. Klevorick, R. R. Nelson, S. G. Winter, 1987. Appropriating the returns from industrial research and development. Brookings Papers on Economic Activity (3), 783–820.

MacCormack, A., J. Rusnak, C. Baldwin, 2004. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. Harvard Business School Working Paper 05-016.

Marti, D., 2002. Use binary-only kernel modules, hate life. LinuxJournal.com, 06/19/2002. http://www.linuxjournal.com/article.php?sid=6152.

McGaw, A., 1987. Most wonderful machine: Mechanization and social change in Berkshire paper making, 1801-1885. Princeton, NJ: Princeton University Press.

Meyer, P. B., 2003. Episodes of collective invention. Working Paper 368, Bureau of Labor Statistics, U.S. Department of Labor, http://www.bls.gov/ore/pdf/ec030050.pdf.

Morrison, P. D., J. H. Roberts, E. von Hippel, 2000. Determinants of user innovation and innovation sharing in a local market. Management Science 46(12), 1513-1527.

Nuvolari, A., 2001a. Collective Invention during the British industrial revolution: The case of the Cornish pumping engine. DRUID Working Papers 01-05, Copenhagen Business School.

Nuvolari, A., 2001b. Open source software development: Some historical perspectives. Working Paper, http://opensource.mit.edu/papers/nuvolari.pdf.

Osterloh, M., S. G. Rota, 2004. Open source software development – Just another case of collective invention? Working paper, http://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID561744_code383702.pdf?abstractid=561744&mirid=1.

Osterloh, M., S. Rota, M. von Wartburg, 2001. Open source – New rules in software development. Working Paper Institute for Research in Business Administration, University of Zurich. http://www.ifbf.unizh.ch/orga/downloads/OpenSourceAoM.pdf.

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

Raymond, E. S., 1999. The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary. Sebastopol, CA: O'Reilly.

Russo, M., 1985. Technical change and the industrial district: The role of interfirm relations in the growth and transformation of ceramic tile production in Italy. Research Policy 14(6), 329–343.

Sattler, H., 2003. Appropriability of product innovations: An empirical analysis for Germany. International Journal of Technology Management 26(5-6), 502–516.

Schrader, S., 1991. Informal technology transfer between firms: Cooperation through information trading. Research Policy 20(2), 153–170.

Teece, D. J., 1986. Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy. Research Policy 15(6), 285–305.

VDC., 2004. White paper, Venture Development Corporation. Referenced in: Linux now top choice of embedded developers. LinuxDevices.com. http://www.linuxdevices.com/news/ NS2744182736.html.

von Hippel, E., 1987. Cooperation between rivals: informal know-how trading. Research Policy 16, 291–302.

von Hippel, E., 1988. The Sources of Innovation. New York: Oxford University Press.

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

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

Webb, W., 2002. Pick and Place: Linux grabs the embedded market. edn.com. http://www.reed-electronics.com/ednmag/contents/images/253780.pdf.

West, J., 2003. How open is open enough? Melding proprietary and open source platform strategies. Research Policy 32(7), 1259-1285.

Wichmann, T., 2002. FLOSS final report – part 2: Free/Libre open source software: Survey and study – firms' open source activities: Motivations and policy implications. Berlecon Research. http://www.berlecon.de/studien/downloads/200207FLOSS Activities.pdf.

Appendix

Table 1: Frequency of use of various means to protect code

Means of protection	al-ways	often	some-times	rare-ly	never	N	mis-sing
Revealing only to customers*	7.0%	14.0%	25.6%	11.6%	41.9%	43	17
Revealing only on request of device buyers	5.8%	22.5%	16.7%	16.7%	38.4%	138	59
Revealing only after delay	2.8%	11.4%	21.3%	19.9%	44.7%	141	56
Loadable binary modules	16.1%	19.5%	17.5%	10.7%	36.2%	149	48

* This means of protection applies to software vendors only. Observations with missing data or with a reply "don't know" are not included in the calculation of percentages.

Table 2: Share of code that is revealed

Type of organization	Mean	Median	St. dev.	Min.	Max.	N	mis-sing
Software company	57.5%	60%	35.9%	1%	100%	39	21
Device manufacturer	42.3%	25%	36.3%	1%	100%	69	45
Component manufacturer	58.8%	60%	40.2%	1%	100%	17	6
Commercial, all	*49.3%*	*50%*	*37.3%*	*1%*	*100%*	*125*	*72*
University / non-profit research org.	90.0%	100%	21.0%	30%	100%	20	10
Hobbyist	93.3%	100%	19.8%	25%	100%	31	10
Non-commercial, all	*92.0%*	*100%*	*20.1%*	*25%*	*100%*	*51*	*20*
ALL	*61.7%*	*75%*	*38.4%*	*1%*	*100%*	*176*	*92*

Table 3: Descriptive statistics of explanatory variables used in Table 5 (N = 119)

Variable	**Dummy variable, equal to 1 if …**	**Frequency of "0"**		**Frequency of "1"**	
SizeMedium	firm has 11 to 200 employees	75	(63%)	44	(37%)
SizeLarge	firm has more than 200 employees	83	(69.7%)	36	(30.3%)
PolicyEncouraging	firm has encouraging policy towards revealing OSS code	81	(68.1%)	38	(31.9%)
PolicyRestrictive	firm has restrictive policy towards revealing OSS code	108	(90.8%)	11	(9.2%)
TypeSWFirm	firm is a software firm	81	(68.1%)	38	(31.9%)
TypeCompMan	firm is a component manufacturer	103	(86.6%)	16	(13.4%)
AgeAtStart	the difference between the company's age and its experience with Embedded Linux is above the median	56	(47.1%)	63	(52.9%)

Variable	**Explanation**	**Minimum**	**Maximum**	**Median**	**Mean**	**St. Dev.**
CompYearsEL	Number of years firm has experience developing embedded Linux	1	10	4	3.647	1.911
Reason_ Development	Importance (-2 … 2) of reasons to reveal code related … to development	-1.93	2	0.8	0.728	0.821
Reason_Marketing	… to marketing	-2	2	0	0.059	0.848
Reason_Reputation	… to reputation	-2	2	1	0.776	0.926
Reason_GPL	… to GPL (i.e., the license requirements)	-2	2	1	1.149	0.938

Table 4: Correlation matrix of explanatory variables (N = 119). Only correlations with p < 0.1 are shown.

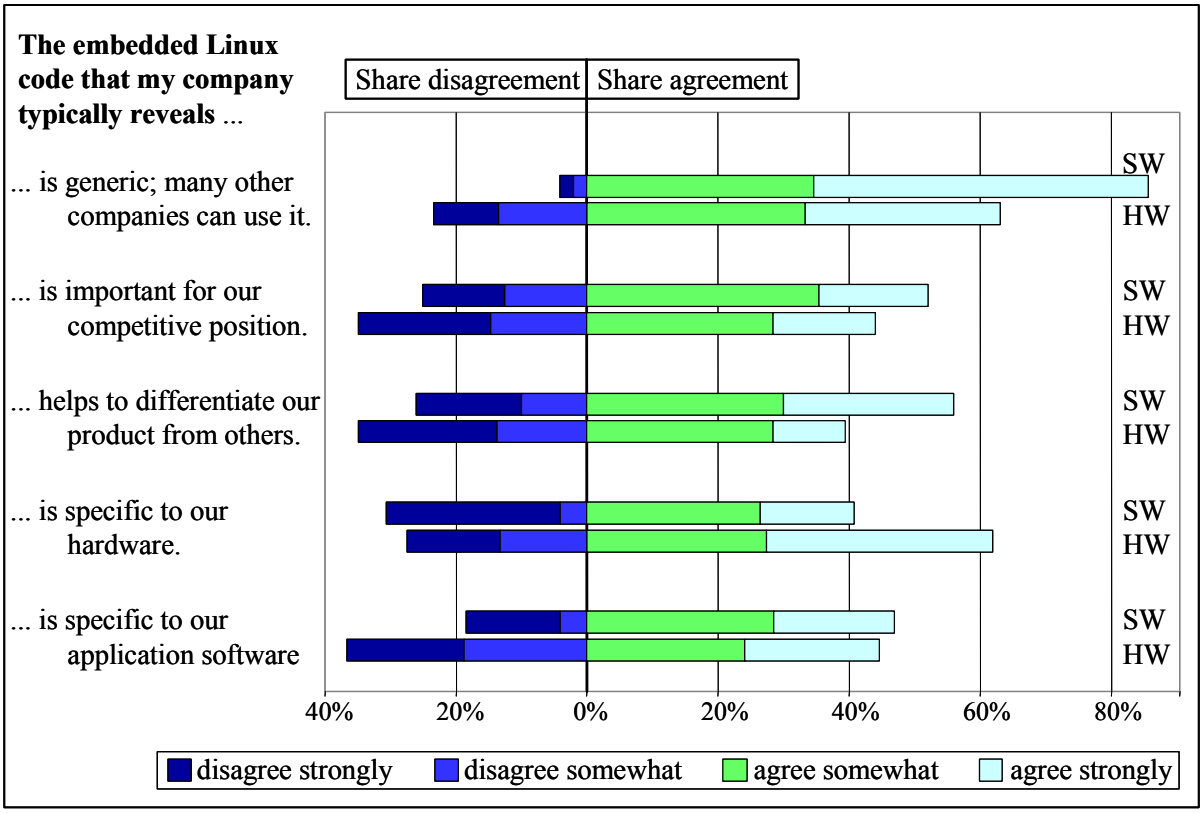
	Size Medium	Size Large	Policy Encouraging	Policy Restrictive	Type SWFirm	Type CompMan	Comp YearsEL	AgeAtStart	Reason Development	Reason Marketing	Reason Reputation	Reason GPL
SizeMedium	1.000											
SizeLarge	not meaningful	1.000										
PolicyEncouraging		-0.176* (0.055)	1.000									
PolicyRestrictive		0.295*** (0.001)	-0.156* (0.090)	1.000								
TypeSWFirm		-0.176* (0.055)	0.188** (0.041)		1.000							
TypeCompMan		0.223** (0.015)			-0.270*** (0.003)	1.000						
CompYearsEL					0.165* (0.073)		1.000					
AgeAtStart		0.584*** (0.000)	-0.257*** (0.005)	0.185** (0.044)	-0.365*** (0.000)	0.322*** (0.001)		1.000				
Reason_Development		-0.157* (0.088)	0.240*** (0.009)	-0.317*** (0.001)					1.000			
Reason_Marketing		-0.238*** (0.009)	0.173* (0.060)	-0.251*** (0.006)	0.223** (0.015)			-0.213** (0.020)	0.358*** (0.000)	1.000		
Reason_Reputation		-0.299*** (0.001)		-0.206** (0.025)				-0.211** (0.022)	0.335*** (0.000)	0.556*** (0.000)	1.000	
Reason_GPL												1.000

* significant at 10%; ** significant at 5%; *** significant at 1%; significance level p in parentheses

Table 5: Multivariate analysis of share of code that firms reveal

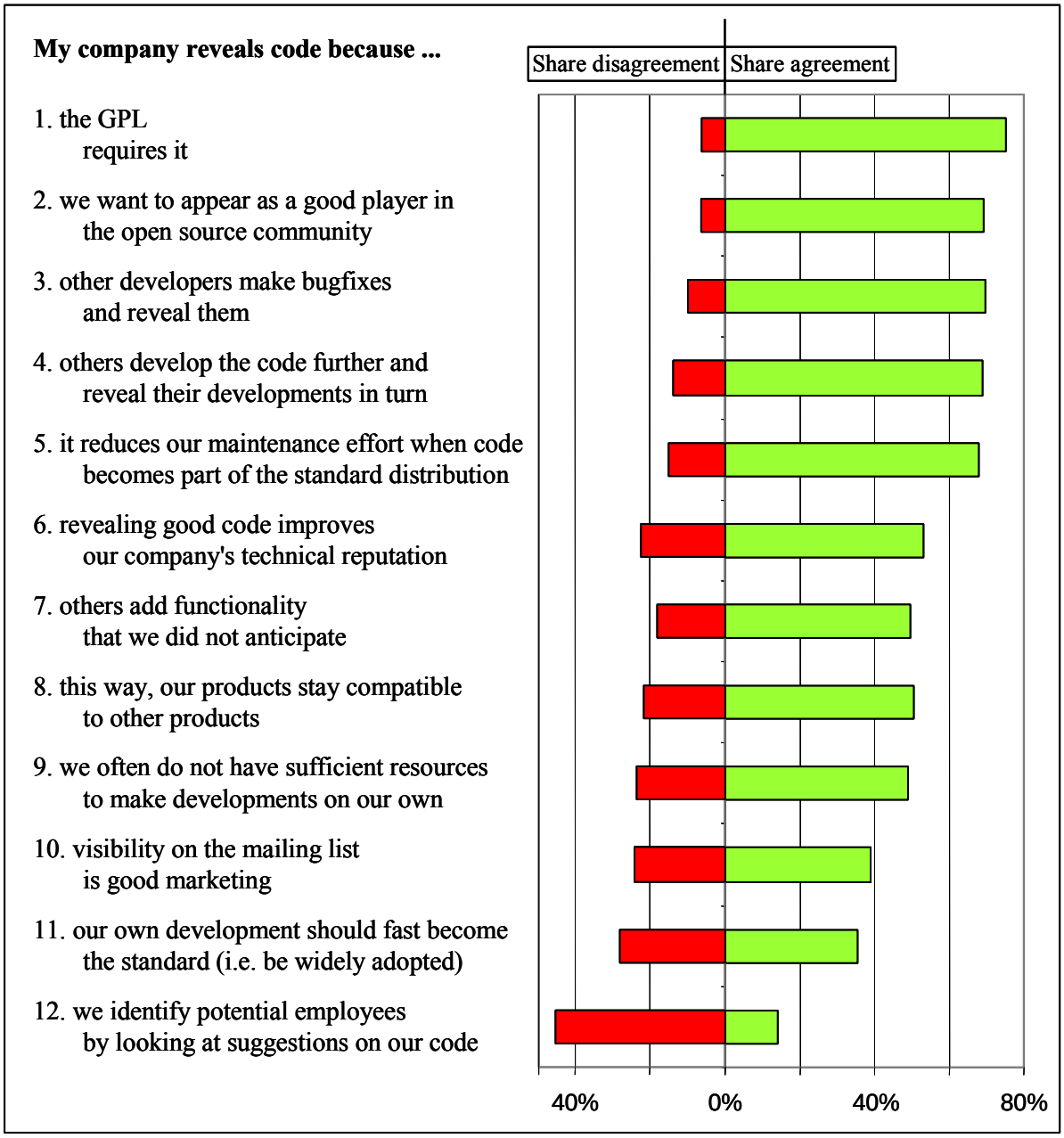
	Tobit		Ordered Probit		Tobit		Ordered Probit	
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
SizeMedium	-3.256 (9.461)		-0.031 (0.253)		-2.681 (9.025)		-0.008 (0.253)	
SizeLarge	-29.780** (12.487)	-27.139*** (10.028)	-0.965** (0.390)	-0.936*** (0.337)	-22.692* (12.016)	-22.257** (9.630)	-0.810** (0.402)	-0.860** (0.340)
PolicyEncouraging	-2.664 (8.113)		-0.077 (0.243)		-6.376 (7.937)		-0.149 (0.247)	
PolicyRestrictive	-23.833* (12.871)	-23.433* (12.787)	-0.923** (0.381)	-0.903** (0.377)	-13.732 (12.738)		-0.645 (0.423)	
TypeSWFirm	19.050** (8.735)	19.259** (8.621)	0.518** (0.248)	0.514** (0.245)	18.989** (8.482)	18.777** (8.262)	0.552** (0.245)	0.501** (0.243)
TypeCompMan	29.113** (11.669)	28.880** (11.649)	0.759** (0.346)	0.754** (0.348)	25.396** (11.347)	25.555** (11.318)	0.669* (0.350)	0.652* (0.338)
CompYearsEL	4.233** (2.001)	4.052** (1.965)	0.143** (0.061)	0.139** (0.061)	3.883** (1.940)	3.391* (1.903)	0.135** (0.061)	0.121** (0.059)
AgeAtStart	19.543* (10.258)	19.006* (9.664)	0.549* (0.293)	0.553** (0.279)	18.908* (9.840)	19.171** (9.300)	0.572* (0.305)	0.588** (0.286)
Reason_Development					8.605* (4.797)	8.787* (4.473)	0.257 (0.166)	0.264* (0.150)
Reason_Reputation					10.042** (4.659)	9.985** (4.209)	0.305** (0.149)	0.278* (0.147)
Reason_Marketing					-0.998 (5.186)		-0.087 (0.163)	
Reason_GPL					1.848 (3.757)		0.063 (0.124)	
Constant	31.030*** (9.827)	29.029*** (8.891)			14.864 (12.003)	14.093 (9.344)		
Observations	119	119	119	119	119	119	119	119
Pseudo R-squared	0.02	0.02	0.082	0.082	0.03	0.03	0.108	0.100
Likelihood ratio (Tobit) / Wald test (Probit)	$\chi^2(8)$=25.4 p=0.0013	$\chi^2(6)$=25.2 p=0.0003	$\chi^2(8)$=25.6 p=0.0012	$\chi^2(6)$=25.4 p=0.0003	$\chi^2(12)$=37.2 p=0.0002	$\chi^2(7)$=34.9 p<0.0001	$\chi^2(12)$=43.3 p<0.0001	$\chi^2(7)$=28.1 p=0.0002
σ (Tobit) / cuts (Probit)	38.40	38.41	0.30, 0.58, 0.92, 1.33	0.33, 0.61, 0.95, 1.37	36.38	36.71	0.81, 1.10, 1.47, 1.91	0.76, 1.05, 1.41, 1.84
* significant at 10%; ** significant at 5%; *** significant at 1%; standard errors in parentheses (for Ordered Probit, robust standard errors)								

Figure 1: Type of code that is revealed



Note: Shares of agreement and disagreement shown for software and hardware firms (see last column). The number of valid responses varies between 48 and 50 for software firms and between 109 and 113 for hardware firms. Responses "neither agree nor disagree" are not shown.

Figure 2: Share of respondents working for hardware manufacturers (N = 137) that agree / disagree to various reasons to reveal



Note: The number of valid responses varies between 105 and 113. The remainder is either missing or the answer was “don’t know”. The share of respondents answering “neither agree nor disagree” is not shown.

Acknowledgements

I am grateful to Carliss Baldwin, Paul David, Marc Gruber, Dietmar Harhoff, Georg von Graevenitz, Eric von Hippel, two anonymous reviewers and to participants at the Workshop on User Innovation and Open Source Software in Munich, the OWLS workshop in Oxford, and a lunch seminar at Harvard Business School for helpful comments and discussions. I thank Mark Tins for valuable help in data collection. All remaining errors are mine.