Research Policy 40 (2011) 576–587

Contents lists available at ScienceDirect

Research Policy

journal homepage: www.elsevier.com/locate/respol

Open source in the firm: Opening up professional practices of software development

Bertil Rolandsson[a,*], Magnus Bergquist[b], Jan Ljungberg[b]

[a] University of Borås, Department of Educational and Behavioral Sciences, Allegatan 1, SE-501 90 Borås, Sweden
[b] University of Gothenburg, Department of Applied Information Technology, SE-412 96 Göteborg, Sweden

ARTICLE INFO

Article history:
Received 5 March 2010
Received in revised form
20 September 2010
Accepted 18 November 2010
Available online 21 December 2010

Keywords:
Open source software development
Professional programmers
Tensions
Strategies

ABSTRACT

Opening up firms to open source has changed professional programmers' work in software development. In their work practice they must cope with two modes of software production: one based on proprietary, closed work situations, the other built around open source community ways of cooperation and knowledge sharing. In this article we present a study of how programmers cope with the co-existence of an industrial/commercial and a community/commons based mode of production. We analyze how they develop strategies to handle tensions that arise from contradictions between these two modes, and how it changes programmers' approach towards open source software development in the company. The study covers proprietary companies that have gradually incorporated open source software (hybrid companies) and SMEs entirely built around open source business concepts (pure-play companies). Four strategies are elaborated and discussed in-depth: Engineering in the lab, Market driven tailoring, Developing the community consortium and Peer-production. At a more general level, the study contributes to our understanding of how the transformation of proprietary production processes into a more open mode of knowledge work is not only associated with company strategies, but also with tensions and new demands on how work is strategically handled by knowledge workers.

1. Introduction

Open source software (OSS) has over the course of a few years changed large parts of the software industry, constituting an alternative or complement to proprietary software, which enables radically new business models, organizational forms and commons-based platforms as new foundations for value creation (Fitzgerald, 2006; Krogh and Spaeth, 2007; Ågerfalk and Fitzgerald, 2008). This broader use of open source software in firms has made employed programmers increasingly exposed to a multitude of open source practices, values and norms (Lin, 2006; von Hippel and von Krogh, 2003). Sometimes this is the result of companies adopting an open development model, as a parallel to their traditional proprietary model of production, and sometimes it is the result of a more ad-hoc practice of problem solving in various development projects. Regardless of the reason, the employed programmers are affected by conditions that can be described as a new and networked mode of knowledge work (Boltanski and Chiapello, 2005; Castells, 2000).

Previous research has focused on motivational issues and questions concerning why programmers in general choose to engage in community-based open source software development (Bates et al., 2002; Bergquist and Ljungberg, 2001; Ghosh et al., 2002; Hars and Ou, 2002; Hertel et al., 2003). A major focus of attention has been programmers' motives for spending unpaid time for developing such software (Bonaccorsi and Rossi, 2003; Krishnamurthy, 2006). Furthermore, there are studies on how to bridge differences between firms' economic and technical interests, and social and ideological motives for taking part in open source communities (Bonaccorsi and Rossi, 2006; Lin, 2006). The focus thus is on how firms combine company and open source community interests (Dahlander and Magnusson, 2008; Fitzgerald, 2006), and how they relate to community based on-line peer-production systems (Benkler, 2002; Lin, 2006). Dahlander and Magnusson (2005) for instance distinguish different relationships between companies and open source communities by claiming that several companies tend to keep their involvement in open source development to a minimum without breaking open source norms, what they call a commensalistic relation to the open source world. Fitzgerald (2006) also describes a broader development in which free and open source software is integrated into a commercially viable form, i.e. what is referred to as OSS 2.0 (Fitzgerald, 2006; Ågerfalk and Fitzgerald, 2008).

* Corresponding author. Tel.: +46 731508501; fax: +46 334354006.
E-mail addresses: bertil.rolandsson@hb.se (B. Rolandsson), magnus.bergquist@ituniv.se (M. Bergquist), jan.ljungberg@ituniv.se (J. Ljungberg).

0048-7333/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.respol.2010.11.003

However, there is a lack of studies addressing what these changes mean to professional programmers working in different organizational settings. By focusing on the organizational level, previous studies do not capture the effects on the organization from the individual programmers' perspective, who have to handle the changes in their daily work practice while developing open source software in a company context. To these programmers, the expansion towards openness in professional software development is not just a continuously emerging OSS 2.0 phenomenon but is also characterized by tensions between open source and proprietary models of organization and business (Lin, 2006; Ven and Verelst, 2008). The focus for this study is therefore on employed programmers, who in their daily work handle tensions between closed proprietary and open modes of production. These modes represent different value systems and principles for organizing programmers' work, which in different settings will have different impacts on programmers' work practices. Values of openness, forms of organization and practices, as well as technical demands associated with the open source movement, are re-interpreted by the programmers in relation to the organizational conditions for software development within a business context. The research question is formulated as follows:

What tensions are generated by the co-existence of the proprietary mode and the community based open source mode of software production, and what strategies do employed programmers develop to handle them?

This question addresses issues on what openness means when open source software becomes part of daily work and how organizational conditions regulate the way programmers organize their use of open source software. Such issues are important in order to understand changing work conditions for employed programmers as well as a broader change towards openness for knowledge workers and knowledge work in general. Furthermore, by focusing on the level of the individual programmers' work practice, the study sheds light on important conditions for firms' opportunities to realize their intentions with open modes of software development.

The paper is structured as follows. In Section 2 we give the background for how open source has developed from being mainly a community-based form of software development to become part of a commercially oriented software strategy, forming a new mode of production. Section 3 covers data collection and method. Empirical data are presented in Section 4, followed by an analysis and discussion in Section 5, while a summary of the results and the conclusions drawn are set out in Section 6.

2. Open source in the firm

Open source software (OSS) as a phenomenon has gone from being mainly an issue for ideologically driven developer communities to becoming a serious alternative to proprietary software for many companies and public organizations. Since the late 1990s, firms have increasingly engaged in open source communities, seeking to combine community and company interests while enabling their employees to capture value created outside their own organization (Dahlander and Magnusson, 2005; O'Mahony, 2003; Ven and Verelst, 2008). This diffusion of open source to a commercial context has been described as a transformation of open source software practices into a more mainstream form, aligned and integrated with proprietary software practices, in what has been dubbed 'OSS 2.0' (Fitzgerald, 2006; Ågerfalk and Fitzgerald, 2008).

One example of this transformation is the spread of less restrictive licenses. The core ideological values (e.g. viewing programs as communal resources) as they were coded in the Free Software definition (GNU's Bulletin, 1986) are inscribed in the GNU General Public License (GPL) (Himanen, 2001; Szczepanska et al., 2005). Since GPL prevents proprietary appropriation of work in the public domain, software with this license would be less attractive to commercial actors that want to make proprietary derivative works (O'Mahony, 2003). To make it easier for open source and proprietary software to coexist, a plethora of less restrictive licenses have become widely used, as well as the dual licensing strategy i.e. the same software is offered under two different licenses (Fitzgerald, 2006; Välimäki, 2003, 2005). The spread of open source has on the one hand served to reduce the significance of the ideological aspects emanating from the open source movement, but on the other hand also contributed to the diffusion of free and open source values in business oriented settings. Values such as the appreciation of programming skills, sharing of solutions, helping others, learning for its own sake, voluntary cooperation, as well as status and reputation achieved through community recognition (Bergquist, 2003; Bergquist and Ljungberg, 2001; Scozzi et al., 2008; Stallman, 1992; Stewart and Gosain, 2006; Raymond, 1999) are still relevant but may be reinterpreted in a business context.

Another part of the transformation of open source is its gradual spread from horizontal to vertical software domains. The lion's share of available open source software has for a long time been found in horizontal domains as infrastructural software such as the Linux operating system and Apache web server, where requirements and design issues are largely part of established programmer wisdom (Fitzgerald, 2006). Due to this, the first movers towards incorporating open source software in traditional proprietary software firms were large hardware-focused vendors such as IBM, HP, and Sun. Also the early "pure" open source firms like Red Hat and Mandrake were built around services related to infrastructural software. Now that open source has started to spread to vertical domains such as business software (e.g. ERP-systems), requirements analysis and design have become crucial, which previously was not at the forefront of open source.

2.1. Two modes of production

The transformation of OSS has altered the very nature of the software industry, both by changing the development process at the supply side, and by giving an alternative to the traditional choice of building or buying at the demand side (Fitzgerald, 2006). This has led to a situation where firms with a proprietary tradition have to relate to community managed open source projects (Lin, 2006). Because community managed open source projects take on large market shares, proprietary firms are forced to relate to them. Open source is also increasingly seen as a strategic move for proprietary firms that want to become more innovative. Similarly, firms built around open source business models are forced to relate to proprietary companies in order to develop their business, since they act on the same market.

Firms mainly built around a pure open source business model, use a community oriented mode of developing software, while most firms with a proprietary tradition incorporating open source software, methods and practices have not entirely replaced previous proprietary and closed practices. Instead, parts have been incorporated into existing methods and models, sometimes leading to a combination of traditional industrial software production and community based open source software production (Feller and Fitzgerald, 2002). Thus, we here have two different types of firms with roots in two different modes of software development. In the following sections we will refer to these modes as an industrial/commercial and a community/commons based mode, understood as two different logics that are partially not compatible; i.e. they are institutionalized modes of understanding how resources can be organized in a powerful and legitimate way despite incompatibilities (Boltanski and Thévenot, 2006). In the industrial/commercial mode, the participants in a software project are employed developers following the norms and practices of proprietary software production in their organizational context. In the community/commons based mode, the participants are peers that often contribute on a voluntaristic basis following the norms and values embraced in open source communities.

The industrial/commercial mode naturally unfolds in the two ideal types of organizations: hierarchy and market (Coase, 1937; Tilly, 1999; Williamson, 1975). Hence, production is based on employees who follow management commands within hierarchic production or what Coase (1937) would call the firm. Still, market logic is also relevant to these employees since they may find themselves coping with constraints caused by the external market (e.g. customer demands). This mode can be described as a secular approach to open source software development, in which open source motives and ideology are regularly questioned due to constant demands on rationalization. With reference to the German sociologist Max Weber, professional programmers can be described as part of a business context demanding that they engage less in faith (i.e. open source ideology) and more in calculations of what type of technology they need in order to achieve planned goals (Weber, 1978, 1992). The programmer will face an industrial logic, in which the mode of production resembles efficient machinery within the organization. However, goals in a business context will also be marked by market logic, meaning that open source peer-production and values can be questioned due to a focus on the firm's capability to handle external competition. Open source values and practices could in that context still become an instrument for optimizing market shares or creating new sales opportunities.

The community/commons based production mode resembles what could be described as civic community logic, emphasizing informal and personal relations, transparent peer-production and software as common goods (Boltanski and Thévenot, 2006). It relates to a third ideal-typical organizational component, singled out as an alternative to hierarchies and markets, namely networks (Podolny and Page, 1998). Networks consist of complex, egalitarian, and informal power relations. Contrary to employees holding formal functions, network members are described as personally engaged in managing community identities, norms and activities (Podolny and Page, 1998; Tilly, 1999). In strategies where networks play an important role, personal or less formal ties will make individuals inclined to stress the importance of cultivating open source communities and values e.g. by sharing code. The type of organizing and production process that is represented by open source projects have also been likened to a bazaar (Demil and Lecocq, 2006; Ljungberg, 2000; Raymond, 1999), i.e. a marketplace where people enter and leave, sell, buy and exchange goods. The characteristics of the bazaar is that actors are not coordinated by price mechanisms (as in markets), formal hierarchies (as in firms) or strong ties/long relations (as in networks); there is no selection of members or contracting parties; and there is no definitive delimitation of roles between users and producers and no enforced work roles (Demil and Lecocq, 2006). *Commons-based peer production* has been suggested by Benkler (2006) to describe this production mode. Benkler also depicts commons-based peer production as an alternative to Coase's (1937) mentioned market and firm relation, were production is organized by individuals seeking out arenas to be creative in, while following a complex set of motivations.

2.2. Tensions between open and proprietary practices

When firms open up their software development to open source and to open source communities (Chesbrough, 2003), they thus expose their programmers' way of working to a multitude of open source practices, values and norms. By doing so employees are enabled to capture, innovate and capitalize value created outside the company (Fitzgerald, 2006; O'Mahony, 2003; Ven and Verelst, 2008). At the same time, their work is also opened up to networks shaped by a civic community logic, emphasizing informal and personal relations, transparent peer-production and software as common goods (Boltanski and Thévenot, 2006; Podolny and Page, 1998). Firms may also become obliged to contribute to value creation outside the firm, where the appropriation of invested resources are out of their control (Chesbrough and Appleyard, 2007; Dahlander and Magnusson, 2005). This is due to logic of legitimacy in which software is seen as something that should be public, which can complicate intentions to organize an internal mode of software development in accordance with an industrial or market logic.

A set of tensions may then emanate from the fact that programmers have to manage different logics driving the dominating software development mode within firms and communities. Finding themselves in-between these two modes, the employed programmers develop individual strategies for governing their capabilities and how they organize their work, i.e. arranging different forms of self-governance (Miller and Rose, 1995). How these strategies take shape gives important information about how professional programmers handle software development opened up to open source practices. It also helps us understand how attempts to improve the dynamics and the ability to innovate in the firm are conditioned by the interplay between organizational requirements and professionals' software development practices. Professionals' strategies could be defined in relation to individual motives and interests in open source communities (Bonaccorsi and Rossi, 2006). However, in organizing their software development strategies, programmers need to take into account whether the strategies will be perceived as effective and legitimate by their colleagues and how they fit into the overall organizational context. Good relations with open-source communities may enable individuals to improvise and innovate in their daily work, but their professional strategies will still be conditioned by the organizational context in which they work. At a practical level this may affect how openness is treated in terms of formal policy, intellectual property regulations, open source licensing forms as well as norms and culture, internally and among customers, partners and competitors.

As a consequence, programmers must put their knowledge and mode of work under constant scrutiny. In firms where efficiency and rationalization are predominant reasons for adopting an open mode of software development, programmers obviously could be inclined to question or even reduce community values and open source contributions. A strategy restricted to utilizing open source communities, in the same manner as Dahlander and Magnusson (2005) refer to when talking about commensalistic relations between firms and networks of open source programmers, could then take shape (Ven and Verelst, 2008). Likewise, programmers could react against such an instrumental approach by articulating strategies stressing open source ideology and the importance of cultivating open peer-production and information sharing. However, by organizing a professional practice in which it becomes common to debate standpoints and solutions in a wide set of networks stretching outside the firm, further reflections and continuous adjustments in how open source is perceived will also be enabled inside the firm. Programmers will be able to widen their base for making use of open source and amalgamating contradicting interests and norms. Instead of just achieving objective and instrumental knowledge demanding that professional open-source programmers devalue community ideals, professional strategies enabling different ways of utilizing and/or cultivating open source communities at work become possible.

3. Method

In order to analyze how professional open source programmers handle different tensions between what is described above

Table 1
Type of companies and number of interviews performed in each company.

Hybrid companies		
Hybrid 1	Large global and service oriented company in the telecom sector	6 interviews
Hybrid 2	SME developing sensor technology and real time systems for public transports	2 interviews
Hybrid 3	Large global and service oriented company in the defense industry	3 interviews
Hybrid 4	SME Vehicle service provider	2 interviews
Hybrid 5	SME Enterprise Resource Planning (ERP) developer and vendor, newly acquired by global enterprise	2 interviews
Pure-play companies		
Pure-play 1	Consultants with customer oriented approach (adapting applications, service level agreements, education, support)	4 interviews
Pure-play 2	Developer and vendor of office systems	2 interviews
Pure-play 3	Developer and vendor of Content Management System (CMS)	2 interviews
Pure-play 4	Consultants with customer oriented approach in CMS and web	4 interviews
Pure-play 5	Open source service platform and application provider	2 interviews
Pure-play 6	Open source security solutions	1 interview

as an industrial/commercial mode and a commons/community based mode of software development, interviews have been done with 30 programmers from two types of companies; pure-play and hybrid (Feller and Fitzgerald, 2002; Lin, 2006). *Hybrid companies* are companies with a history in proprietary oriented and industrial/commercial mode of software development, that have gradually incorporated open source software and methods. Fifteen interviews were done with programmers working at hybrid companies. The other fifteen interviewees were recruited from so called *pure-play companies* that are formed around the idea of creating a business model from open source. These companies were often entrepreneurial and service-oriented small and medium sized firms (SMEs), started by programmers with experience of commons/community based software development who wanted to approach commercial software development based on open source code, methods and practices.

Fitzgerald (2006) points to the fact that many hybrid firms so far have been engaged in infrastructural software development, which historically often has been large hardware oriented companies. Most of the selected companies in the hybrid category in our study are hardware oriented, in line with the general development. To increase the variation of firms in our sample, we included interviewees working for smaller hybrid SMEs, one of which also was software oriented. A similar approach was used for the selection of pure-play companies. The majority of chosen companies are consultancy and service oriented firms from which we selected interviewees with a varied range of approaches to open source practices and methods. In keeping with the approach for hybrid companies, a more hardware oriented example was selected to get a broader coverage of different approaches within the pure-play category of companies. Table 1 gives an overview of the types of companies and number of interviews included in the study.

The selection of programmers for interviews was made more difficult as a result of the ambiguity connected to the term “programmer”. The definition of the term ‘programmer’ as used in this study is rather wide and covers designers, coders, system developers, software engineers, software architects and to some extent project leaders. The interviewees also differed in other aspects, such as age and length of employment. Programmers who worked for hybrid companies with a long proprietary history, frequently were qualified engineers (with university degrees) whereas those who worked for pure-play SMEs had a wider variety of educational backgrounds with university degrees from different educational programs, stretching from primary school teaching to engineering. We actively sought to select interviewees to balance gender bias and the fieldwork had an explicit agenda to capture women’s work situation. However, it was difficult to find female programmers, which is a general experience from similar studies (Cuckier, 2003; Hafkin, 2003). The study finally ended with interviews with six women professionally engaged in open source software development. They were found both in hybrid and pure-play companies.

The main empirical data is derived from semi-structured qualitative interviews that lasted 1–2 h each. We did the interviews at each informant’s work place. Some interviews were for practical reasons done with several people in the same interview. We sought to assure that all individuals had equal opportunities to explain their view on their specific work situation. The objective with this approach to interviews was to gain a deeper understanding of the interviewee’s view on different subjects, to understand the underlying logic of behavior rather than the behavior itself. A set of open-ended questions based on topical themes where created before the interviews. Questions were asked to capture the process of introducing open source into the company and how this affected the programmers work practice. The situated character of a qualitative interview should be stressed (Miller and Glassner, 2004). Even if the informants talk about organizational and structural issues, they do this from a situated perspective reflected by their role in the organization, which means that this becomes a part of their work practice forming the way they will relate to open source.

It was not decided prior to the study exactly what should be included in the activities associated with open source. Instead the interview questions were open for the interviewees to describe their use of open source in different organizational contexts. This resulted in varying definitions of what is included in open source. Some interviewees stressed the code while others pointed at juridical definitions of code and software, or the wider spectrum of organizational forms and practices associated with open source, such as peer-production, network based organizing and collaboration. The rationale underlying the choice of companies and informants was to collect data covering different contexts. The varied organizational contexts studied allowed the identification of different types of strategies that programmers use while trying to cope with ideological tensions due to the intersection of open source in their professional practice (Kvale, 1996).

The software Tams-analyzer (http://tamsys.sourceforge.net) was used to code and analyze the empirical data. Interviews were read and re-read systematically before being coded into categories. By alternating between theory and content, the categories were developed. Codes that were both theoretically anchored in concepts like hierarchies, markets and networks and empirically grounded in our interviews were in this way distinguished and used in the analysis (e.g. content of work, pragmatism, openness). Also logical dependencies between different categories were searched for. Example of dependencies found were, for instance, relations between programmers, the company and the open source movement (Ragin, 2000).

It is sometimes argued that a general methodological problem in qualitative analysis is that persons interviewed are not entirely consistent in their arguments. Instead they tend to move between different discursive positions. However, on the level of analysis these divergences are seen as variants of strategies that can be used by a programmer. The aim with this qualitative analysis is not to draw definite conclusions about the general impact of open source code on software development (Klein and Myers, 1999), but to analyze how programmers cope with socially constructed and context dependent tensions. Still, the approach allows both explorative and theoretically related conclusions to be drawn about the ways programmers manage such tensions (Miles and Huberman, 1994). This is also the reason why empirical data is presented as it is. We have chosen to split the results and discussion sections and to include rich accounts from the interviews in the results section allowing developers to elaborate on tensions and strategies, thus providing the reader with contexts for the different positions taken in the interviews

4. Results

Below the results from the study are presented, organized under three headings that capture general themes in the empirical data. These themes are a result of the initial data coding process. Under work *context*, the focus lies on what characterizes the main activity of the programmer in terms of software development, e.g. technology as a goal, or as a means to achieve something else. *Values of openness* present norms and ideologies related to open source as well as proprietary models and how these are used, combined, contested and developed. *Regulations of work* capture perspectives that inform about institutional regulations and constraints. The two types of companies selected for the interviews – hybrid and pure-play companies – will be compared in relation to the three themes.

4.1. Programmers in hybrid companies

4.1.1. Work context

In hybrid companies, programmers' work was based on a proprietary approach to software development. Open source software had gradually been included as part of the development process. The interviewees also referred to a context of work and technology marked by firms acting on an international market, of which many had a hardware oriented focus; i.e. software was frequently seen as part of hardware products. This often meant that developers were occupied by operating complex systems and adapting hardware solutions to different conditions. In one case (Hybrid firm 5), programmers were also working for a software oriented SME developing Enterprise Resource Planning (ERP) systems.

With the exception of those who worked for this latter firm, most of these programmers did not develop software for end users. Furthermore, none of the programmers engaged in co-production or in any other contacts with the customer that could affect their work directly. Although, the awareness of customer needs was more pronounced in hybrid SMEs, most programmers working with open source code in hybrid firms were still more engaged by the development of software inside well-defined project groups. They focused on opportunities within their project groups to achieve technical excellence, stability, security, effective memory and processor use, scalability, and other domain specific demands on the system such as up-time, remote control, climate conditions, built in control of performance etc. Those who worked for the SME that developed ERP-software described a less defined and more experimental use of open source, which also addressed customer needs more directly. Still, the work was done within their group.

The programmers motivated their engagement in open source with an interest in quality and in being able to select tools that were adjusted to specific technical needs. A slight difference can be detected between the hybrid SMEs and the larger companies. In the SMEs, open source communities were more directly providers of code that was used to facilitate and speed up the programmers' work, especially in the early stages of the projects. Larger hybrids were not as dependent on inflow of tools and knowledge from communities. However, in both types of companies programmers still tended to work in-house in different project teams, focusing on technological problem-solving. Work within these groups was informal and autonomous. Still, decisions on the nature of problems to be solved were taken in other parts of the organization.

4.1.2. Values of openness

Within the context of hybrid companies, programmers described openness and access to the source code as something desirable. To them open source and openness was not a matter of politics, but rather associated with improved quality and control over the software developed, as well as with a sense of becoming innovative and of being on the cutting edge of technological development. In the following quotation one respondent describes open source as something advanced, exciting and constantly up to date, which engaged them but also demanded skills and gave good conditions for developing both innovative products and the individuals' expertise:

> I believe that the best thing with this work is that we have the opportunity to work to a relatively large extent with technologically advanced stuff. This is what I call great fun. I think it is great to work with new technological gadgets, as well as working on problem-solving on a more complicated higher level, where you also are able to get down to a level which is concrete and closer to the hardware. This is something I really enjoy. When looking at my work, I guess working with this type of problem-solving really is my biggest interest. [Hybrid 1]

References to a "higher level" of technology in the quote above demonstrate how work with open source was perceived as advanced. They more or less saw themselves as researchers solving difficult problems. Some programmers even described their work place as a lab in which they built prototypes, and explored new technological possibilities relatively far away from the customer. They appreciated an environment in which they became part of a team of technological experts, sharing expertise and code with each other. It should be noted that most programmers were engineers with a university degree – working for industrial companies, producing both hardware and software – who over the years had built up an extensive capacity to refine and develop new technology. However, when given the chance to use open source tools and code all programmers – even those working for the software oriented hybrid SME – referred to what can be described as a scientific autonomy in how they brought on and tried to solve different technological problems. Open source as a tool, they argued, provided them with the latest, transparent and most advanced components, enabling them to push their knowledge further while facilitating their work with software development. A programmer from one of the hybrid SMEs described his appreciation of open source as a way to gain control:

> Since the development of the Linux kernel is so rapid it is good to be able to follow that change and simultaneously adapt and change ones own modules. We add our functionality and have high demands on performance, in terms of memory use and other performance issues, which makes it much easier to create the modules we need. [Hybrid 2]

Summing up: by making the programmers less dependent on software vendors contracted by management, open source was seen as a tool helping them to both catch up with a rapid technological change and to improve their autonomy while changing the code. Open source supported the programmers' autonomy while they developed both their own competence and software. Against these pro-openness values stood a concern for slow and bureaucratic decision-making processes in their firms, complicating their opportunities to take advantage of open source code. Programmers in large hybrid firms mainly referred to a necessary judicial procedure based on the company's need to protect intellectual property rights, but which sometimes was directly hindering openness and innovation. This created a tension between companies' need for control of intellectual assets and programmers' relations to communities outside the firm. The companies were dependent on innovations that could be stimulated by access to open communities. Still the programmers were hindered by the companies' need to control proprietary claims or future business opportunities in the form of patents, which e.g. a GPL-license would jeopardize.

4.1.3. *Regulations of work*

Most of the programmers interviewed appreciated openness but were well aware of license issues and organizational constrains for sharing code with communities. All hybrid companies had some kind of policy or regulation for how to act in relation to communities and for how open they could be with the code. In two large hardware oriented firms programmers described how open source development was explicitly regulated by a clearing document that had to be filled in by every programmer who was considering using an open alternative in a system or application. The document was cleared by the patent department, and if no risk was identified, the programmer could choose the open alternative. This formal procedure indicates a rather distant relation to open source communities:

> I guess we haven't contributed back that much - I know that this was an issue before, when we still belonged to this other company. There were some contributions at that time, but I think we are relatively restrictive today. *Is that because of security reason?* I would say so, because everything that our programmers do is part of the company, and we want to hold on to that. It is one thing if someone plays around at home, but in our company I would say that we are rather reluctant as regards contributions. [Hybrid 3]

The quoted interviewee came from a hardware oriented firm where not just proprietary business but also security reasons demanded that they were careful with direct activities in open source communities. In addition, all programmers had a rather pragmatic strategy for managing this tension between proprietary demands versus openness. They all saw their work as a contribution to cutting edge technological development, in which access to open source software and documentation made an important resource. By using open source code they got access to an environment where different software components were developed in a rather intensive pace, as well as discussed and tested by many co-workers. Still, they did not see their work as a contribution to the open source movement. Instead, open source was perceived as a resource that made it possible for them to contribute to a wider technological development that was disseminated through the company's channels, which everyone would profit from in the long run. Consequently, these programmers never contributed to community based software development. The reason put forward was that their potential contributions were so special that they would be useless for other programmers. By using open source code, they could as professional programmers identify themselves with something desirable, but they were mainly interested in problem-solving capabilities able to deliver high quality solutions. Programmers working for hybrid SMEs shared this pragmatic or problem-solving approach, but as the following quote demonstrates a problem solving strategy then also became associated with ideas of meeting customer needs:

> You may choose to buy the code from someone else. But the advantage is that instead of generating 2000 hours work there are so many good developers out there to rely on, helping you to reduce it to 200 hours. One may turn to these environments, where the original developers or firm run a project where the code is donated. In this way the code is continuously developed, and the customer can pay for the work we do rather than for being locked up by agreements. [Hybrid 5]

What is revealed here is an attitude where open source ideology is supported in principle, but this does not result in code actually being given back to the community. Instead, the "winner" is the customer who gets the benefit of a rapid open source development processes. A programmer working for a large hardware oriented firm where customer needs were less pronounced also referred to principles by stressing the importance of being "a fair player when it comes to open source" [Hybrid 1]. Nevertheless, to a vast extent the use of open code was a matter of increasing efficiency by reducing working hours while still keeping high quality standards. The main reason for using open source code was that it made it easier for the programmer to contribute to reliable high quality software development in house. Another respondent from one hybrid SME also made clear that the advantages of taking part of quality code and ideas in an open source context were subject to certain conditions. In particular, it demanded responsibility and some sort of prudence in how code was selected:

> Even if I don't know whether it has to do with open source, I guess it becomes a bit different. The fact that you are testing stuff that you have found on your own, means that you have a lot of responsibility to select the code that fits your tasks the best - as long as it does not differ to much from what the others are already doing. [Hybrid 4]

Some ideological tensions can be identified here. Generally, developers in hybrid firms supported an improved sharing of and access to the source code. They described themselves as problem solving researchers working in labs, who considered that the use of open source demanded both technological expertise and strengthened autonomy, e.g. by reducing constrains from agreements and licenses. Programmers working at hybrid SMEs also stressed that open source could speed up their software development. At the same time, the sharing of source code and their autonomy regularly had to be strategically constrained by the company, even if this sometimes lead to formal, slow and bureaucratic procedures of work. Especially those working for the bigger hybrid companies saw the tension between their need to affiliate with open source and the need to guard intellectual property rights. Even if the programmers appeared to be focusing on problem solving in their work rather than dealing with the ideological problems of the open source movement, they still had to relate to tensions between openness and juridical constraints

4.2. *Programmers in pure-play companies*

4.2.1. *Work context*

Programmers who worked for pure-play firms described a great variation in work context, values and regulation of work. They often depicted their context and content of work as focused on software products and services, but in one pure-play firm the programmers also talked about developing software for different hardware oriented customers. Many of the selected companies developed and adapted web applications around open source solutions, e.g. web, CMS, ERP and workflow systems, and added different services as additions to applications or as pure consultant services. Only a few of the pure-play companies developed their own technologies.

Pure-play developers often had a close contact with customers and the market. They approached their customers in various ways and used open source as a way to create alliances with the customers. Generally, their work with software development appeared to be informal and autonomous, both on an individual basis and in project teams, and contrary to former hybrid companies none of them received predefined problems delegated within a formal hierarchic organization. Instead they developed activities in which customer or open-source community preferences meant that they constantly reorganized their mode of developing software.

4.2.2. Values of openness

All programmers working for pure play companies put a high value on openness, the sharing and contribution of source code, and how it supported their opportunities to be autonomous experts. Compared to programmers in hybrid companies we found a wider variety of ways in which they related to open source communities. For instance, pure-play companies 1, 4 and 6 were mainly focusing on web-applications and consultant services. Several programmers in these companies were relatively instrumental in their relation to communities. Even if they promoted the use of open source and often wanted to associate themselves with the principles behind the movement, these programmers did not necessarily want to become an active part in open source communities. Open source projects were instead used as a raw material for tailoring software and business offers to customers. In accordance with the business idea in these companies, to add value and build a business model around open source code, they wanted to create services that took advantage of open source systems or parts of applications. Even if sharing was seen as something positive, and licenses demanding free access (such as GPL) were preferred, they also declared that they could choose proprietary alternatives if the customer opted for it.

Many of these programmers had a background as engaged open source contributors or advanced users. Working for an open source oriented SME was for them a way to turn a previously private interest into professional work. Some of them participated in open source communities on a private basis and thus shared this interest with their colleagues. They also regularly expressed a direct relationship to the open source movement and its ideology. In some cases, they had applied for their present job because they were attracted by the company's open source profile:

> No, I would never work with any other sort of code. It's not that I have anything against working with certain types of commercial products, but I have to think about what technologies will be relevant within two or three years. I don't want to invest time in something that I don't believe in. After all, it's all about what you want to work with, to find a technology that you feel comfortable working with. [Pure-play 1]

Even if a strong belief in open solutions was expressed, no-one advocated a non-commercial approach. Still, a tension was identified between what appear as a purist approach – pro open source in its most non-compromising form – and the kind of demands that were put on the company as an actor in a service and customer oriented competitive business. This tension was addressed by attaching a high level of importance to an honest and pragmatic approach both to community-driven open source development and to the customer. This pragmatism was crucial for how to make sense of professional work with open source code in a commercially driven company. Accordingly, some of the programmers claimed that one of the most important advantages of open source was that they could avoid sales pitch. Using open, already existing systems meant that focus could be put on creating value for the customer as well as advocating the benefit and usefulness of open source. This position was perceived as compatible with the ideologically driven open source development where many of them also had their background.

Programmers who worked for pure-play companies with a more active role in open source communities then considered themselves as highly capable of influencing the open source software development they engaged in and saw it as a part of their job to contribute with solutions. They also identified themselves with their companies and described themselves as persons devoted to work, with difficulties to separate private interests from the professional role. In some cases they also had founded and remained project owners of a community on which they based their open source business, making it difficult to distinguish work, community and leisure. Private projects could coexist with professional projects in different parts of the community.

Contrary to those who saw open source as an opportunity to tailor software in accordance with customer demands outside the communities, these programmers stressed the importance of looking at the development going on within their community and argued that the customers had to adapt their needs to the community agenda. This, it was argued, was best for the customers in the long run since it provided them with the best software quality. However, when it came to actual contributions of code, programmers who collaborated closely with a community also had two different and distinct ways of understanding tensions between interests of the company and of the community, coinciding with how the company organized their engagement in their respective communities.

The first example is mainly found among programmers who worked for a company formed around an open source initiative and a product targeting a small specialized and globally diverse customer segment. In this case, open source became a business strategy used to create a critical mass around the product, which was free to download and use. However, if the users wanted to influence the development of the product, they (preferably companies) paid a membership fee allowing them to contribute with code. Besides having the possibility to influence the development of the product in favor of own interests, this meant that customers could establish a good reputation through the community as leading innovators. This community had attracted large multinational companies of which many had interests in patents, and the company that owned the project acted as a community host contributing with code and facilitating their members' opportunities to take part. To the programmers in this company, this also meant that their community mainly became an efficient work tool for spreading the code they developed. They were engaged in a community mainly consisting of companies who were willing to share code, but also wanting to make money out of their engagement. Licenses were chosen to avoid viral effects, which imply a strategy to handle tensions between company interests and an ideology of openness. This rather pragmatic strategy can be illustrated by one of the programmer's founder stories:

> It was when we decided to reconstruct the company; we chose to become open source. It was a mix of honest open source ideology and a way to make a living. We were not perfectly honest because we had a strategy to give away the system for free. It's not that uncommon. Many companies do that. You give something away so you can sell a system. We thought it through a lot before we decided to go for this model. We also chose a license model that later would fit with a subscription system. So, the idea was to separate the two. [Pure-play 5]

This should be compared with the explicit ideological tensions found among respondents working in two other firms, organizing their software development by using conventional communities rather than multinational companies. These firms mainly developed enterprise and content management systems, and while doing so the respondents could sometimes see a need to combine closed and open source code. However, these programmers also saw themselves and the professional work they did as part of a movement contributing to a wider change in society. Developing code in a community was not just seen as a part of doing business but also as a political project, in which it was important to show that they could both volunteer to offer competence in the struggle for something good as well as profiting from doing that. They wanted to demonstrate that it was possible to combine ideological and instrumental interests associated with taking part in a community based open source development:

> I put a lot of time into choosing the right projects. I mean, I do have a clear strategy for what I am doing. It has to be a *worthy* project, a project that gives something back. It is important to contribute somehow. I prefer to work on finding flexible models in terms of financial compensation, as long as I feel that it gives something back. [Pure-play 2]

A careful selection of worthy projects implies that the quoted interviewee above had a just or even moral reason to his/her use of open source. But, the search for projects enabling such noble contributions did not emerge as an easy task. Being asked about the tension between business and ideology, one of the programmers emphasized that it was difficult to handle such tensions and that open source “gurus” who only argue for ideological goals underestimated the proportions of the problem. Another respondent from the same company described how important it was to have control over the project portfolio to fully integrate economic compensation for work with a more ideologically driven motivation to achieve something good for mankind.

4.2.3. Regulations of work

Compared to the respondents who worked at hybrid companies, most programmers from pure play firms were not that focused upon internal conditions for technological problem solving in their work. Programmers who saw open source as a raw material for tailoring software, described how they worked on an individual basis at the customer’s site, and tackled their ideological tensions by emphasizing an identity that related to clients’ needs. Unsuccessful attempts to set up firm specific communities were mentioned, and instead their work was conditioned by their capacity to come up with customer adapted software solutions and high quality support. For them the advantage with open solutions was described as improved opportunities to work closer with customers. It was seen as an affordable and efficient mode of producing high quality software together with their clients. Instead of starting a new project by writing lengthy specifications, open source made it easier to co-operate with the customer using prototypes and iterations.

In this way, ideological issues of sharing, free access to source code and quality issues related to specific open source software development methods became intertwined with a client and business oriented rationality. By being able to use freely available open source code in a commercially driven customer oriented project, the developers would not primarily share code, but improve their co-operation with the customer and thereby focus directly on generating value from open source. In this context, absence of proprietary claims was seen as a reason to why trust could much easier be established between the company and the customer. As advocates of open source code they did not have to defend the solutions as had it been an in-house developed product. Hence, they claimed that it provided them with opportunities for a more autonomous objective and critical role, and therefore also believed that they would achieve both higher quality in their offerings and an honest face towards the customer. Several respondents pointed to the fact that an open source environment also would secure that software continued to exist independent of the company’s existence. Utilizing open source became a way to both enforce honest competition and empower the customer:

> Every programmer here is more or less into open source as a way of thinking. It is a clean and smooth way of delivering things to the customer who gets full rights to do their own thing with the code. They see that you are honest and don’t try to fool them and deliver crappy stuff. [Pure-play 4]

This way of associating an open mode of work with a trustworthy customer relation can be compared to the programmers who worked for other pure play companies directly involved with different open source communities. Informal and problem focused collaboration in the community then became the prototype for how software development was conducted within the company – mainly in connection with test driven agile-methods and pair-computing. In addition, there was a difference between the programmers who worked for the company approaching open source as a business strategy and those who worked at firms in which the community were ideologically anchored, in how they co-ordinate their work with their communities. In the first case, the community was a means used to create a critical mass around the development of the product. The core of the community mainly consisted of customers, such as industrial companies paying a membership fee enabling them to influence the development of software in favor of their own interests. The customers, who frequently were multinational firms with interests in different patents, became a part of the community and thereby had an impact on how the programmer worked. In this type of community, which can be described as a business oriented open consortium, open source ideology and topics like licenses rarely became an issue.

In the second case, however, the community-based approach generated a clear tension between customer focus and public good. These programmers tried to handle the tension between what the customer wanted and what the community needed. Mainly, the open source communities were then described as their own personal networks, and the customer as an external actor who had to be convinced to subordinate internal interests in favor of the development of the community:

> We manage to attract many contracts with clients, which has as a consequence that work and community activities merge. In practice I do some community work paid by our customers, for example customizations that can be brought back to the community. We write agreements with the customers to make sure that everything we do for them also benefits the community. [Pure-play 3]

It should be noted that even if the customer still played an important role in these firms, the quoted respondent above talks about agreements that would limit their impact. Furthermore, a need for regular negotiations with customers is described. Some other programmers also tried to develop a strategy to avoid customer involvement although customer orientation was seen as the company’s strategic goal. The focus was rather on the community and community activities merging with what the quoted programmer referred to as work.

5. Discussion

The results section illustrated different consequences of open source becoming a part of companies’ software development strategies. On the one hand, consequences are described in previously proprietary companies that have become hybrid, with both proprietary and open software development models. On the other hand, we also describe pure-play companies that have emerged, built around the idea of creating viable business models emanating from open source software and open source communities. Part of our results confirm previous research claiming that firms fuel a more commercial approach towards open source (Fitzgerald, 2006; Ågerfalk and Fitzgerald, 2008). However, from the programmers’ point of view – which is the subject of the article – we may also

Table 2
Programmers' strategies in relation to their firm context and to communities.

Firm context	*Customer*	Market driven tailoring	Developing the community consortium
	In-house	Engineering in the lab	Peer production
		Utilizing	*Cultivating*
		Relation to Community	

claim that this development is associated with significantly different strategies for how they relate to community/commons-based production or industrial/commercial production (Benkler, 2006; Stam, 2009). These are professional strategies that may co-exist within the firms, extending programmers' ability to act by enabling different combinations of market- and hierarchic demands within the firm with norms and values of open source communities (Lin, 2006; Ven and Verelst, 2008).

Previous research has drawn a line between an industrial mode of production versus a newer community based mode of peer-production (Benkler, 2002; Demil and Lecocq, 2006). Still, when focusing on programmer activities we identified four major strategies related to tensions between different organizing principles in the firm and the community respectively – *Engineering in the lab*, *Market driven tailoring*, *Developing the community consortium*, and *Peer production*. Some of these strategies were based on open source as a *utilizing* device for the programmer in a company, whereas other strategies aimed at *cultivating* community-based open source software development. Further aspects shaping these strategies were whether they were *customer* or *in-house* oriented. Based on these analytical distinctions a subset of strategies used by the programmers were identified (Table 2).

It should be noted that in some contexts it could be difficult to make a clear distinction between strategies building on the utilization of open source versus strategies building on the cultivation of open source. Utilization commonly refers to using open source without giving back to the community, whereas cultivation builds on mutual benefits for both the developer and the open source community. It has been pointed out that utilizing open source as a part of a company's brand also can be a way to cultivate the open source movement in general as it accredits the use of open source among professionals and firms, and thereby helps strengthen the movement (O'Mahony, 2003). However, when analyzing programmers' strategies it is important to acknowledge these differences as sources for tensions that will affect their strategies. It is necessary to make a distinction between utilizing or cultivating community resources to understand how the strategies become meaningful to the studied programmers. The four strategies presented in the table above are discussed in following sections.

5.1. *Engineering in the lab*

Engineering in the lab as a strategy was described as "problem-solving research", dealing with well defined problems distributed to the lab by managers sitting somewhere else in the hierarchy (Coase, 1937). The programmers defined themselves as skilled engineers, who had developed a general interest in technology, and believed that open source would improve opportunities to achieve better technical solutions. This strategy was common among programmers employed by hybrid companies. Many of these hybrid firms were engaged in hardware and patent dependent infrastructural software development and the programmers organized their work in a way that fostered a technologically driven culture distant from customers.

The programmers recognized open source values and ideals as admirable and saw themselves as problem solving engineers or experts contributing to a common good technological development. Open source was framed as a provider of high quality code and as advanced development environments, but the programmers were not devoted to the idea of openness as such. Any environment offering high quality code and similar technical benefits would support the contribution to a positive and more general technological development. Thus, the main strategy was to make use of open source code because it suited their demands for handling complex but predefined tasks, and not because they wanted to change the world in accordance with the ideals of the open source movement. Even though this strategy is not directed towards community work it bears resemblance to a typical logic found in open source communities often described as "scratching a personal itch" (Raymond, 1999): software development in open source communities often start with a programmer having an itch, i.e. a problem that needs to be solved, which becomes the starting point for a new development project. Similarly the engineers in the lab oriented themselves towards open source because they found tools and solutions that helped solving technical problems that they on a general level shared with other programmers.

This strategy was challenged by the company's internal organization and juridical considerations. A tension is created in the borderland between programmer's technical needs and the regulative impact of a formal hierarchy (Benkler, 2002; cf. Coase, 1937). As a consequence the strategy creates possibilities to make autonomous design choices based on technical preferences, at the same time as it demands that they cope with the internal structure of the company and its juridical constraints. Open source provides opportunities to fulfill personal needs and goals in the process of innovating work in relation to locally defined peers, which also includes utilization of resources provided by the community, rather than cultivating open source communities. Engineering in the lab as an identity project is about becoming aware of how to exploit functions within a rather formalized hierarchy (Podolny and Page, 1998; Tilly, 1999). Accordingly, the strategy focuses on work in project groups, aware of the fact that sharing only goes on internally among peers in the company.

5.2. *Market driven tailoring*

Market driven tailoring is a customer oriented strategy often associated with the development of software wrapped in different services. The strategy is based on programmers that are in direct contact with the market, having customer interaction with the goal to tailor software and services according to customer needs. The tailoring strategy was followed mainly by programmers who work in pure-play SMEs using open source as a foundation for their main business model, mostly when developing web-applications and engaged in consultancy.

The strategy refers in some sense to open source demands on freedom of information as a common good (Castells, 1996). Nevertheless, for these firms open source was an important strategic asset in order to compete, and the programmers were engaged in trying to make this approach their own strategy. A tension was identified between open source ideals and commercial demands, and the strategy was to claim that the customer became less dependent on one supplier when choosing open source instead of proprietary solutions. Open source was justified by referring to the importance of sharing and how open source could work as a market strategy for building trustworthy customer relations (Podolny and Page, 1998). Open source was perceived as a high quality input helping programmers to solve problems defined by customers, rather than as an environment to cultivate in its own right.

Despite a certain awareness among programmers about customer related judicial issues, this strategy did not refer to patent departments or formal documents used when needed to get permission to handle open source code. A more extrovert approach could be found, less preoccupied with problems distributed within a formal and hierarchic organization. In a sense these programmers can be seen as parts of a broader bazaar organization (Raymond, 1999), making them less inclined to rely on in-house procedures for defining tasks and controlling code (as was the case with Engineering in the lab). Instead, tailoring implied constant negotiations with the customers, which also meant that work was regulated in relation to a commercial logic (Benkler, 2006; Coase, 1937). A recurring argument was that the customer only should pay for software tailored in accordance with their organizations' specific needs, not the development of the basic functionality, which was the case when buying from a proprietary firm. The business idea for the company harmonized with the strategies of the programmer, and in comparison with the previous strategy the tailoring strategy was externally oriented towards software and business propositions being created close to the customer, on an informal and individual basis.

5.3. Developing the community consortium

Developing the community consortium was a strategy particularly articulated by programmers who actively cultivated open source communities as a way to develop applications and service platforms for other, often hardware oriented firms. This strategy invokes an indirect propagation of open source values by taming ideological tensions between community and commercial interest, through building an exclusive community together with customers who pay fees to be able to contribute to the software development. The company is placed right inside the community (Benkler, 2006). Such a strategy resembles both the characteristics of a consortium and a community, i.e. more planned and less emergent than a pure community. Contrary to the former cases where programmers utilized and certified the quality of open source code (Podolny and Page, 1998), either within an internal hierarchy or as a strategic advantage to communicate to customers, a consortium is a way to actively cultivate an open source arena in which developers can share and contribute to a community while negotiating the content of the code (Raymond, 1999; Zeitlyn, 2003).

In this way, the quality of the code is guaranteed by the community consortium that develops the software. However, it also extends the community frame by including the customers. As such, it becomes a community that is a means for marketing and attracting customers to take part in the software development process. Being a small company and acting on a global market can explain the need for such a strategy, which is founded on the idea of fostering a community of peers around the world who also are customers for the products. In accordance, the strategy promotes licenses facilitating combinations of open source and proprietary code (Fitzgerald, 2006).

5.4. Peer production

Finally, Peer production as a strategy is, like the tailoring strategy described above, founded on open source ideas of freedom of information (Castells, 1996; O'Mahony, 2003). It can be described as community cultivation, which articulates "classic" open source values that associates with a peer production approach to software development that will guarantee a quality of code that all involved profit from in the long run. This strategy includes business demands, but still emphasizes the importance of achieving something that is a common good. It therefore demands that specific customer needs must be secondary. In accordance, tensions between community and business interests are handled by prioritizing the community and it's capability to develop high quality software that as many as possible will profit from in the long term.

The Peer production strategy is mostly used by programmers who take an active part in open source communities. In our data they are represented by programmers in pure-play SMEs involved in developing enterprise and content management systems. The strategy is founded on the expertise of the community, which also regulates the work. A hierarchy is created in which the community overarches the relation to the customer. Cultivating the community becomes the ultimate goal, and business and customers are means to achieve this (cf. Benkler, 2006; Demil and Lecocq, 2006). Therefore, the quality of open source software should first of all be guaranteed by the review process and the informal hierarchy that exist within the open source community, and then be used to solve customer problems. This is also a community strategy where personal networks of developers play a key role. However, the priority of the community also means that the programmers must persuade customers to subordinate their immediate needs in favor of the joint effort of the community and trust the expertise within the open source communities they themselves are part of. The close relationship with the customer is similar to the tailoring strategy, but the goal is the opposite: instead of tailoring software to the customer, the customer should be tailored to the software, based on the logic that community driven development will favor the software – and thereby the customer – in the long run.

5.5. Programmer strategies in between two modes of software development

Previous research argues that open source software development nurtures a new mode of community/commons-based peer-production that has come to challenge a traditional industrial/commercial mode of production (Benkler, 2002). Individuals are then said to seek out ideologically driven communities to be creative in, described as bazaars coordinated by a logic of openness and legitimacy stimulating an innovative software development (Demil and Lecocq, 2006). There is research also indicating that open source (e.g. product domains, business strategies and product support) rather is becoming a crucial part of commercial software business – a change Fitzgerald refers to as OSS 2.0 (Fitzgerald, 2006; Ågerfalk and Fitzgerald, 2008).

Our results describe how the professional programmer relates to both these modes while working with open source. By looking at how they in their profession handle open source software, we may say that their work practices are indeed opened up to community/commons-based peer-production, but that their daily work also demands that they strategically handle tensions between open source development and proprietary business models. They are part of open source inspired business models that try to incorporate the ideologically driven commons-based spirit, but still operate within a context based on market demands. The tensions created by this situation oscillate between community driven software development and a proprietary industrial commercial mode of production. Rather than just contributing to a development either towards what Dahlander and Magnusson (2005) describe as a commensalistic relation to open source, or to the introduction of more routines within a hierarchic industrial mode of production, our results show how the programmers develop strategies motivated by different benefits of using open source as means to support flexible and autonomous professional practices. Their different strategies create possibilities for both capturing value created in various software communities, and coping with different ideological tensions that are constrained by firms' internal organization and relation to the market.

6. Conclusion

In this paper we have presented a study of programmers' strategies for coping with tensions emanating from the co-existence of an industrial/commercial and a community/commons based mode of production. We have identified four different strategies among the programmers that can be summarized as follows:

- The first strategy is to handle tensions between openness in the communities and bureaucracy in the organization, by utilizing open source as engineers and instrumentally focusing on formal hierarchic conditions and opportunities to experiment with in-house projects.
- A second strategy is to look upon open source as something that could be utilized to tailor software according to customers' demands. Tensions between market and ideological demands on openness are handled by empowering the customers through access to the source code.
- The third strategy is to cultivate a community consortium by involving users as exclusive business partners paying a membership fee. Tensions between community and business are handled by combining a user/customer discourse, allowing a limited set of customers to become privileged users/developers with the right to shape the development process.
- Finally, the fourth strategy is to cultivate an open source community of peer-producers, where tensions between community life and customer demands are handled by constantly persuading customers to conform to the joint expertise of their community.

These strategies indicate that, if companies wish to successfully harness the possibilities offered by open source, they must be fully aware of the conditions under which open source enters the organization (cf. Krogh and Spaeth, 2007). Previous research, focusing upon the relationship between firms and communities, has concluded that a creative and profitable inflow of ideas will not happen spontaneously (Dahlander and Magnusson, 2008). Based on our study we can add that when firms aim at improving their mode of software development by opening up to co-existence with open source communities, they have to recognize that the programmers will face communities built on a different logic than traditional proprietary modes of software development. Opening up the organization for open source mindsets and work practices will put new demands on the existing organization (Stam, 2009). From a programmer's perspective, open source is then a possibility, but not without constrains. It has to be nurtured by the organization and actively handled by the programmers to become the innovative force it is claimed to be. The developers that are studied in this paper offer strategies for how to cope with some of the tensions that emerge when these two modes of production meet. The fact that they both apply to a range of businesses, suggest that their strategies also may help us understand how other contemporary knowledge workers can capture ideas and value generated outside company borders (Benkler, 2006; West and Lakhani, 2008). However, before we draw such general conclusions further research is needed in other fields of production, were similar practices are emerging.

Acknowledgement

The research presented in this paper has been funded by The Swedish Council for Working Life and Social Research (FAS), project nr 2007-0673: Open Source and the Changing Professional Practice of the Programmer.

References

Ågerfalk, P.J., Fitzgerald, B., 2008. Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy. MIS Quarterly 32 (2), 385–409.

Bates, J., Di Bona C., Lakhani, K., Wolf, B., 2002. The Boston Consulting Group Hacker Survey.

Benkler, Y., 2002. Coase's Penguin, or, Linux and The Nature of the Firm. The Yale Law Journal 112 (3).

Benkler, Y, 2006. The Wealth of Networks: How Social Production Transforms Markets and Freedom. Yale University Press, New Haven, CT.

Bergquist, M., 2003. Open source software development as gift culture: work and identity formation in an Internet community. In: Garsten, C., Wolff, H. (Eds.), New Technologies at Work. People, Screens and Social Virtuality. Berg Publishers, Oxford.

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

Boltanski, L., Chiapello, E., 2005. The New Spirit of Capitalism. Verso, London, New York.

Boltanski, L., Thévenot, L., 2006. On Justification: Economies of Worth. Princeton University Press, Princeton.

Bonaccorsi, A., Rossi, C., 2003. Why open source software can succeed. Research Policy 32 (7), 1243–1258.

Bonaccorsi, A., Rossi, C., 2006. Comparing motivations of individual programmers and firms to take part in the open source movement: from community to business. Knowledge Technology and Policy 18 (4), 40–64.

Castells, M., 1996. The Rise of the Network Society – The Information Age; Economy Society and Culture, vol. 1. Blackwell Publishers, Massachusetts.

Castells, M., 2000. Materials for an exploratory theory of the network society. British Journal of Sociology 51 (1), 5–24.

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

Chesbrough, H., Appleyard, M.M., 2007. Open Innovation and Strategy. California Management Review 50 (1), 57–76.

Coase, R.H., 1937. The Nature of the Firm. Economica. New Series 4 (16), 386–405.

Cuckier, W., 2003. Constructing the IT skills shortage in Canada: the implications of institutional discourse and practices for the participation of women. In: Proceedings of SIGMIS Conference, Philadelphia.

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

Dahlander, L., Magnusson, M.G., 2008. How do firms make use of open source communities? Long Range Planning 41 (6), 629–649.

Demil, B., Lecocq, X., 2006. Neither market, nor hierarchy or network: the emergence of bazaar governance. Organization Studies 27 (10), 1447–1466.

Feller, J., Fitzgerald, B., 2002. Understanding Open Source Software Development. Pearson Education, Harlow.

Fitzgerald, B., 2006. The transformation of open source software. MIS-Quarterly 30 (3), 587–598.

Ghosh, R.A., et al., 2002. Survey of Developers. Free/Libre and Open Source Software: Survey and Study. FLOSS. Final Report. International Institute of lnfonomics, Berlecom Research GmbH.

GNU's Bulletin 1986. 1 (1), p. 8. Available at http://www.gnu.org/bulletins/bull1 (accessed 5 March 2010).

Hafkin, N.J., 2003. Some thoughts on gender and telecommunications/ICT statistics and indicators. Paper presented at the 3rd World Telecommunication/ICT Indicators Meeting, Geneva, 15–17, January. http://www.itu.int/ITU-D/pdf/5196-007-en.pdf (accessed 5 March 2010).

Hars, A., Ou, S., 2002. Working for free? Motivations for participating in open source projects. International Journal of Electronic Commerce 6 (3), 25–39.

Hertel, G., Niedner, S., Herman, S., 2003. Motivation of software developers in open source projects: an Internet based survey. Research Policy 32 (7), 1159–1177.

Himanen, P., 2001. The Hacker Ethic and the Spirit of the Information Age. Secker and Warburg, London.

Klein, H.K., Myers, M.D., 1999. A set of principles for conducting and evaluating interpretive field studies in information systems. MIS-Quarterly 23 (1), 67–94.

Krishnamurthy, S., 2006. On the intrinsic and extrinsic motivation of Free/Libre/Open Source (FLOSS) Developers. Knowledge, Technology & Policy 18 (Winter (4)).

Krogh, G., Spaeth, S., 2007. The open source software phenomenon characteristics that promote research. Journal of Strategic Informations Systems 16 (3), 236–253.

Kvale, S., 1996. Interviews: An introduction to Qualitative Research Interviewing. Sage Publications, London.

Lin, Y., 2006. Hybrid innovation: the dynamics of collaboration between the FLOSS Community and Corporations. Knowledge Technology & Policy 18 (4), 86–100.

Ljungberg, J., 2000. Open Source as a Modelfor Organizing. European Journal of Information Systems 9, 208–216.

Miles, M.B., Huberman, M.A., 1994. Qualitative Data Analysis: An Expanded Sourcebook, second ed. Sage, Thousand Oaks.

Miller, J., Glassner, B., 2004. The "inside" and the "outside". Finding realities in interviews. In: Silverman, D. (Ed.), Qualitative Research: Theory, Method and Practice. Sage, London.

Miller, P., Rose, N., 1995. Production, identity, and democracy. Theory and Society 24, 427–467.

O'Mahony, S., 2003. Guarding the commons: how community managed software projects protect their work. Research Policy 32 (7), 1179–1198.

Podolny, J.L., Page, K.L., 1998. Network forms of organization. Annual Review of Sociology 24, 57–76.

Ragin, C.C., 2000. Fuzzy-set Social Science. University of Chicago Press, Chicago.

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

Scozzi, B., Crowston, K., Eseryel, U.Y., Li, Q., 2008. Shared mental models among open source software developers. In: Proceedings of the 41st Hawaii International Conference on System Sciences, Big Island, Hawaii.

Stallman, R., 1992. Why Software Should Be Free. http://www.gnu.org/philosophy/shouldbefree.html (accessed 2010-03-05).

Stam, W., 2009. When does community participation enhance the performance of open source software companies? Research Policy 38 (8), 1288–1299.

Stewart, K.J., Gosain, S., 2006. The impact of ideology on effectiveness in open source software development teams. MIS Quarterly 30 (2), 291–314.

Szczepanska, A.M., Bergquist, M., Ljungberg, J., 2005. High noon at OS Corral – duels and shoot-outs in open source discourse. In: Feller, J., Fitzgerald, B., Hissam, S.A., Lakhani, K.R. (Eds.), Perspectives on Free and Open Source Software. MIT Press, Cambridge, MA.

Tilly, C., 1999. Durable Inequality. University of California Press, Los Angeles, London.

Välimäki, M., 2003. Dual Licensing in Open Source Software Industry. Systemes d'Information et Management 8 (1), 63–75.

Välimäki, M., 2005. The Rise of Open Source Licensing: A Challenge to the Use of Intellectual Property in the Software Industry. Turre Publishing, Helsinki.

Ven, K., Verelst, J., 2008. The impact of ideology on the organizational adoption of open source software. Journal of Database Management 19 (2), 58–72.

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

Weber, M., 1922/1978. Economy and Society. University of California Press, London.

Weber, M., 1930/1992. The Protestant Ethic and The Spirit of Capitalism. Routledge, London, New York.

West, J., Lakhani, K.R., 2008. Getting clear about communities in open innovation. Industry and Innovation 15 (2), 223–231.

Williamson, O.E., 1975. Markets and Hierarchies: Analysis and Antitrust Implications. Free Press, New York, NY.

Zeitlyn, D., 2003. Gift economies in the development of open source software: anthropological reflections. Research Policy 32 (7), 1287–1291.