

UNIVERSITETET I OSLO
Department of Informatics
Technology, Organization and Learning

Managing Firm-Sponsored Open Source Communities

A case study of Novell and the
openSUSE project

Master thesis
(60 credits)

Jan Fredrik Stoveland

May, 2008



Abstract

The interest and use of open source software and methodology has gained an increasing amount of commercial attention, and we are currently witnessing that established proprietary software firms are taking a step further by opening their own software projects in an attempt to create firm-sponsored open source communities. Siobhan O'Mahony's research finds that these firms have to handle a tension between openness and control in their product development, but little research has been done to detail *how* this balance is achieved. I have studied the American software company Novell and the openSUSE project, largely guided by an inductive, qualitative approach supplemented by some quantitative methods. In the study I draw upon Niklas Luhmann's theory of autopoietic social systems to create a distinction between the sponsor firm and the sponsored community, and I investigate the mechanisms that hold the two systems together despite their differences in interests. I argue that there are several elements that ensure a tight coupling between the two systems, including the boundary objects situated between them, the shared communication channels and the efforts of the marginal people whom have roles in both systems. A primary contribution to the theory of boundary objects is a distinction between what I describe as supportive-objects and target-objects. I argue that the latter holds a strong motivating power that should be appended to our understanding of individuals' and collectives' motivation to participate in open source software projects. I explore several possible future scenarios for the evolution of firm-sponsored communities, and find that Novell is pursuing a strategy for managing openness in such a community.

Keywords: FLOSS, managing communities, open innovation, boundary objects, marginal people, autopoietic social systems, Luhmann

Preface

This thesis is submitted under the masters program of “Technology, Organization and Learning” at the University of Oslo, which is under the administration of the Department of Informatics. The interdisciplinary program is a collaboration of the departments of informatics, sociology and pedagogy, and emphasizes the role of information technology in organizational change and learning. Although this thesis draws upon knowledge from all domains, the theories I use are weighted towards a sociological perspective. As the masters program has been established quite recently, this publication is one of the first masters thesis’ to be submitted under this program.

I would like to extend my gratitude to several people that have generously provided help and assistance to my work. First and foremost, I would like to thank my advisor Lars Risan at the University of Oslo for invaluable support and enthusiasm. Together, we have had many interesting and important discussions, and he has patiently read and commented numerous drafts of this thesis and contributed with a load of theoretical ideas. At the university, I would also like to thank my closest fellow students for keeping the morale up in our library, and our professor Lars Groth for offering assistance and keeping us united.

I owe a great deal of gratitude to Novell and the management at the R&D department in Open Platform Solutions, whom have generously opened their doors and supported my research on their organization. In particular I would like to thank Andreas Jaeger and Michael Löffler for facilitating and accepting my visit in Nuremberg, Justin Steinman and Meike Chabowski for making the introductions, Martin Lasarch and Henne Vogelsang for providing information and showing me around, and Timo Hoenig for being a great friend.

During my 2-year masters program, I spent the spring semester of 2007 at University of California, Berkeley. I would like to thank my professor Neil Fligstein for insights and assistance, Siobhan O’Mahony for supporting my work, Stephen Barley and the colloquium at the Centre for Work, Technology and Organizations at Stanford for inspiration, Matt Asay for taking interest in my work and sponsoring my participation at the Open Source Business Conference, and all my friends at the International House in Berkeley for an amazing experience.

In agreement with my advisor, I have chosen not to include any appendixes to the thesis. Upon request I will nevertheless be happy to provide any underlying data, such as the full results from the survey, cross-tabulations from SPSS, field notes, interview guides, interview transcripts or the source code for my written statistics-software program.

Good reading!

Glossary

This is a collection of words that are frequently used in the open source domain, and that are also used several places in this thesis. Some words are familiar, but may have a slightly different meaning in this context (e.g. "distribution"). The following describes the meaning of the words in the context of this thesis.

Bug – A software bug (or just "bug") is an error, flaw, mistake, "undocumented feature", failure, or fault in a computer program that prevents it from behaving as intended (e.g., producing an incorrect result).

Bug-report – A report detailing a bug in a program. Bug-reports are often collected in an electronic repository (such as with the software program-tool "Bugzilla"). A bug-report may also include a written discussion between the author of the report and other developers about how the bug may be solved.

Distribution (as in 'the openSUSE distribution') - GNU/Linux comes in several 'flavors'. Each 'flavor' is called a Linux distribution (popularly referred to as a 'distro'). openSUSE is one such Linux distribution, as is Red Hat, Ubuntu, Fedora, Mandriva, Debian Linux and many more¹.

Fork - 'Forking the code' happens when developers take a copy of source code from one software package and start independent development on it, creating a distinct piece of software. Free or open source software is, by definition, possible to fork without permission of the original creator. The term may carry a negative connotation in some contexts, as a fork can often be started as a result of a disagreement in a software development project. Sometimes the term "branch" is therefore used instead.

GNU/Linux - An operating system originally authored by Linus Torvalds in 1991 built with GNU software. Has since evolved as the largest open source project involving contributions from many thousand individuals over the world. Linux is a derivative of the UNIX operating system. For simplicity, the "GNU" abbreviation is left out in the thesis.

GPL – GNU General Public License (GNU GPL or simply GPL) is a widely used free software license, originally written by Richard Stallman for the GNU project. It is the license used by the Linux kernel. The GPL is the most popular and well-known example of the type of strong copyleft license that requires derived works to be available under the same copyleft.

¹ This arrangement is different from other familiar operating systems such as Microsoft Windows or MacOS, which only come in one 'flavor' delivered by one company (different release versions such as Win98, WinXP, Vista, etc still only count as being the same 'flavor').

IRC – Internet relay chat. A form of real-time Internet chat or synchronous conferencing. It is mainly designed for group (many-to-many) communication in discussion forums called channels.

Mailing list - a collection email-addresses used to reach multiple recipients and a common way of distributing information and facilitating discussions in open source communities. Subscribing to the list is (normally) voluntary, and any subscriber can post a message that reaches out to all the recipients of the mailing list. The term is often extended to include the people subscribed to such a list, so the group of subscribers is referred to as "the mailing list", or simply "the list".

Package - Short for 'software package'. A software package is a collection of all electronic files that are associated with any type of software program, ranging from desktop applications to modules in the kernel of the operating system.

Packager - A job description for an engineer who's job is to maintain one or more software package for a GNU/Linux distribution.

Patch – a small piece of software designed to update or fix problems with a computer program or its supporting data. This includes fixing bugs, replacing graphics and improving the usability or performance.

Release - Short for 'software release'. Used when a piece of software is officially updated and published (normally with an increased number indicating the release version).

Upstream - Refers to a direction toward the original authors or maintainers of software that is distributed as source code. When a developer wishes to contribute a patch or fix for a bug targeted at being included in (a future release of) the original software, the contribution is sent 'upstream'.

Wiki – A collection of web pages designed to enable anyone who accesses it to contribute or modify content, using a simplified markup language. Wikis are often used to create collaborative websites and to power community websites. The collaborative encyclopedia Wikipedia is one of the best known wikis.

Contents

INTRODUCTION	1
Background	1
Motivation.....	3
The case.....	3
Research questions	5
Structure of the thesis	7
THEORY	9
THE OPEN SOURCE DOMAIN	10
Models of software production	10
A brief history of open source	13
Commercializing open source, open-sourcing commerce	15
Community and firm relations	17
Willingness and resistance.....	19
Firm-sponsored open source communities	21
BOUNDARY OBJECTS.....	23
What is a boundary object?	24
Shared knowledge, mutual identity	25
Translation, negotiation and reconciliation.....	27
Types of objects	31
AUTOPOIETIC ORGANIZATION THEORY	32
Introducing Niklas Luhmann	33
Features of autopoietic social systems.....	34
Interactive, societal and organizational systems	36
Criticism to the theory.....	39
Open source software projects as social systems.....	41
METHODOLOGY	43
RESEARCH STRATEGY.....	43
RESEARCH DESIGN	45
Case study	45
Interviews	46
Ethnography.....	47
Simple statistics	48
Survey.....	48
REFLECTIONS.....	50
Triangulation.....	50
Generalizability.....	51
Reliability	51
Validity.....	52
EMPIRICAL FINDINGS	55
THE STRATEGY.....	56
Going for SUSE Linux	58
Birth of the openSUSE project	59
The result	62

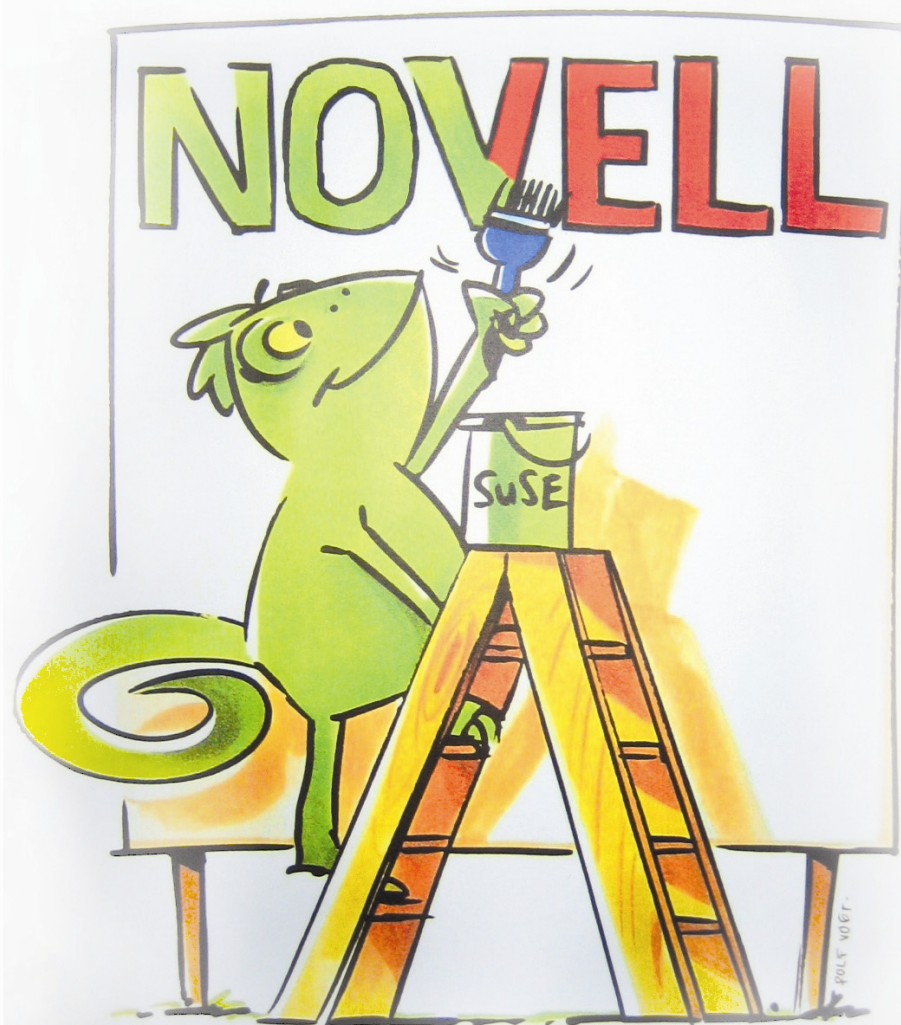
THE SOFTWARE PRODUCT(S)	64
Number of users	64
openSUSE vs SUSE Linux Enterprise	65
THE DEVELOPMENT PROCESS.....	68
Who are the developers?	69
Internal development and the Autobuild system	71
External development and the OpenSUSE Build Service	73
THE OPENSUSE COMMUNITY.....	78
Who is the community?.....	78
Community structure	81
Community activity.....	83
Community contributions	86
Community motivation.....	87
ANALYSIS AND DISCUSSION	91
THEORIZING NOVELL AND THE OPENSUSE PROJECT.....	92
Defining systems.....	92
System connectivity	95
The organizational membrane	96
BINDING SYSTEMS	97
Boundary objects	98
Shared communication channels.....	111
Marginal people.....	113
FUTURE PERSPECTIVES.....	120
Status quo.....	122
Integration.....	122
Extinction.....	125
Divergence.....	126
Failure	128
Bridge.....	128
CONCLUSION	131
Luhmann, Star and Novell.....	131
Exploring theory.....	133
Managing openness	135
REFERENCES	137

List of figures

Figure 1. Novell business units and location of Linux development	4
Figure 2. Motivational aspects of open source development	12
Figure 3. Translation. Modified from Callon (1986).	29
Figure 4. Boundary objects and translations. Modified from Star and Griesmer (1989)	30
Figure 5. Elements of a research project. From Holter & Kalleberg (1996, p.33).....	44
Figure 6. openSUSE release cycle (dates not exact).	66
Figure 7. Communities within communities.....	70
Figure 8. Direction of development - external contributions.....	75
Figure 9. Employee participation on developer lists – users and posts.	85
Figure 10. Motivation for community participation - distribution of frequencies.	88
Figure 11. The Novell organizational system and the interactive openSUSE system	93
Figure 12. Model of a bug-report.....	99
Figure 13. Keywords in the openSUSE object.....	104
Figure 14. Illustration of translations in the openSUSE object. Model derived from Star and Griesmer (1989).	105
Figure 15. Target-object motivation.	110
Figure 16. Network of boundary objects and communication channels.	112
Figure 17. Participation by employees on mailing lists.	115
Figure 18. Internal and external pressure towards separation.....	118
Figure 19. Elements of system connectivity.....	119
Figure 20. Split code base.	126

List of tables

Table 1. Characteristics of the commercial and open source software model.	10
Table 2. Business models with open source software. Modified from Dahlander (2004).	16
Table 3. List of interviews.....	46
Table 4. Distribution of users and posts on developer mailing-lists.	84
Table 5. Main mailing list - opensuse@opensuse.org	84
Table 6. Employee participation on mailing lists	85
Table 7. Contributions by community	86
Table 8. Community-influence on decisions.....	116
Table 9. Future scenarios for the openSUSE project	121
Table 10. Identity between Novell and the openSUSE community	124



Chapter I

Introduction

Background

Ever since Netscape Communications announced their plans to make their Web browser an open source product by establishing the Mozilla project in January 1998, an increasing amount of software companies have been pledging to open source software development (Demil & Lecocq, 2006). This is an interesting observation, as it is initially hard to understand how open source software development can be profitable on the software market. The unique features of open source software involve granting the user permission to study, change, improve and redistribute the software in a modified or unmodified form, and is often termed “free software”. Open source software can therefore be understood as a public good accessible to everyone (Hippel & Krogh, 2003; O'Mahony, 2003). Nevertheless, several scholars have shown various means of appropriating returns with open source software (Bonaccorsi, Lorenzi, Merito, & Rossi, 2007; Dahlander & Magnusson, 2005; Demil & Lecocq, 2006; Lin, 2006; O'Mahony, 2007a), and an increasing amount of business models surrounding open source software are emerging. The escalating commercial interest in open source software development is most likely a result of the success of the open source model, as detailed by Steve Weber (2004).

Twenty years ago the position of open source software in relation to commercial software development was most uncertain, but today open source software is a major part of the mainstream information technology economy. Nearly 40 percent of large American companies use Linux in some form (ibid). The Apache software holds 65% of the web server market. The email transfer and management software Sendmail powers about 4 out of 5 mail servers in the

world. Increasingly, open source software products such as these are running major enterprise applications for large and small corporations alike (ibid, p.6). Furthermore, it does not look like the open source movement is going to die out any time soon; SourceForge.net is the world's largest open source software development web site, hosting more than 100,000 projects and over 1 million registered users². Although open source software on many occasions has proven to be technically superior to proprietary alternatives, the answer to its success is probably more sociological than technical. Open source software is often developed in a public, collaborative manner, as opposed to software that is developed in a proprietary fashion where the program source code is patented and remains within the developer organization. The latter is the most common form of commercial software development today. The open source software development process is based on voluntary participation and voluntary selection of tasks (Weber, 2004), and attracts talented programmers who's participation is primarily based on intrinsic motivation (Hippel & Krogh, 2003). Furthermore, the Internet enables a vast amount of programmers to participate simultaneously on the same project. This makes it possible to mobilize a large amount of "manpower", which also can raise the quality of the product. Eric Raymond, an influential programmer and writer on the open source movement, explains this logic with what he calls Linus' Law: "Given enough eyeballs, all bugs are shallow" (Raymond, 2001). The organizational model of open source can easily seem somewhat chaotic with the lack of formal authority and price mechanisms to provide governance, but it is no anarchy. There are several alternative mechanisms in place to deal with coordination and complexity in projects, for example through norms, licensing schemes, sanctioning, individual incentives and leadership practices that prevent the communities and projects from falling apart (see Weber 2004 for a extensive description of these macro-foundations).

The development in the software market the last decade has shown that "being open source" may bring success, and we are now witnessing a range of different shapes and forms of open source initiatives blending in various ways with proprietary software and for-profit companies. Many large established software companies such as HP, SUN, IBM, Oracle and Novell (to name a few) have positioned themselves over the last years towards embracing open source technology and methodology. On the one hand, previously proprietary companies such as these are supporting existing autonomous open source projects, by contributing manpower and software code improvements. This has been a typical form of collaboration between the commercial and the open source software domain for some time now. In addition, some incumbent software firms have recently taken a step further than simply participating in open source communities and using open source software. Several proprietary software companies have chosen to open up their own software projects in an attempt to *create surrounding open source communities* to support their

² <http://sourceforge.net/docs/about>

own development. This is what O'Mahony and West describe as *firm-sponsored open source communities* (O'Mahony, 2007a; J. West & O'Mahony, 2005), which is the main theme for this thesis.

Motivation

While autonomous open source software communities have received a great deal of empirical and scholarly attention within the last decade, the research on firm-sponsored communities is fairly limited. The exceptions are found in (O'Mahony, 2007a; Joel West & Gallagher, 2004; J. West & O'Mahony, 2005). This thesis therefore aims at exploring this topic in greater detail, and provide a better comprehension of this organizational model of collaboration. In Siobhan O'Mahony's research, she finds that a major difference between firm-sponsored and autonomous communities is that the former need to handle a tension between openness and control in their product development (2007a). Given that developers earn the trust and prove their competence, the autonomous communities can basically offer the same opportunities for everyone to gain rights to commit code and part-take in strategic decisions. For firms, it is more problematic to offer this kind of openness to external community members, due to their need for control of the product upon which their business model relies. *This relationship is equally present in the case of this study.* I am therefore motivated to find out more about how firm-sponsored open source communities are *managed* in practice.

The case

The case for this study is the American software company Novell (est. 1983) and the openSUSE project (initiated in 2005). Novell has a history as one of the largest proprietary software companies within the domain of operating systems. Towards the end of the 90's their potent operating system 'NetWare' gradually fell into obsolescence, and the company was pressured to explore alternative technologies to replace it. At the end of 2003 Novell acquired the open source company Ximian and the German software company SUSE Linux. The latter had been developing a commercial Linux distribution since 1994. The SUSE Linux operating system became the replacement for NetWare, and has made Novell a major company devoted to open source software development. Although Novell still develops a large amount of proprietary software, Linux is now the backbone of their technology portfolio. The new strategy threw Novell into an extensive internal and external transition of the company. For one of my informants, just switching the desktop on the workplace computers was a major change in the company that he remembers well (interview #1, 18.05.07). In half a year the entire organization switched from licensed Microsoft products to OpenOffice and Linux on all workstations within the company – affecting salesmen to engineers. The business organization is illustrated in figure 1 below. The Linux development in the company is done within the research and development

team (R&D) in the Open Platform Solutions unit. This is where most of the research for this thesis is done. Most engineers in the R&D team are located in Nuremberg, Germany, in the offices of the former SUSE Linux company, although the team also has employees in Cambridge/MA, Prague, Provo, Bangalore, Beijing and in home offices. In this thesis I will refer to this sub-organization as *Novell's Linux R&D*. The other units in the figure represent the other (mostly proprietary) software sectors in Novell.

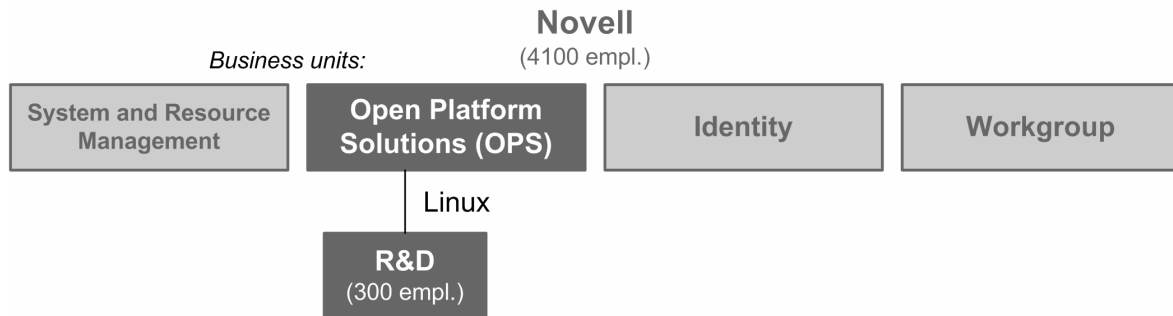


Figure 1. Novell business units and location of Linux development

In August 2005 Novell initiated a project to open up their product development process. Although SUSE Linux had always been an open source product, the development of this software was previously a closed process within the company. The openSUSE project therefore sought to bring development of the Linux product to the outside of the company's boundaries. Today, Novell's Linux R&D work in collaboration with external developers and contributors in the **openSUSE community**, which is Novell's firm-sponsored open source community in O'Mahony's terms. Since 2005, Novell now promotes two Linux distributions³: the **openSUSE distribution** which is developed in collaboration with the external community, and the commercial **SUSE Linux Enterprise** products which are *based on* the openSUSE software. Novell sells licenses and offers support for the latter, and bases many of their other licensed proprietary products upon it.

The community-developed software and the enterprise Linux products are tied closely together. For this reason Novell is struggling with releasing any control and devolving any decision-making rights to external community members. This situation may present a paradox since the motivation among individual community members to participate often relies on their ability to actually influence the software development. Therefore, most informants I spoke with at Novell and in the external community claim that the current status is not a desirable long-term situation. However, the relationship between Novell and the openSUSE community is constantly evolving,

³ See the glossary at the start of the thesis for an overview of some common open source terminology that is used in this thesis.

and even at the time of writing several developments have progressed since the data for this thesis were gathered. As O'Mahony notes, we do not know how this tension between openness and control will transform over time (2007a), and the evolution in Novell is therefore a very interesting case to study.

Research questions

By establishing the openSUSE community, Novell have on the one-hand extended their own organization. At the same time they are keeping the external developers at arms-length by not granting them any direct authority to commit code to the system. This situation poses some interesting questions about what organizational phenomenon we are witnessing: Can we understand Novell and the openSUSE community as one and the same organization, or as two separate ones? Is it even possible to regard the openSUSE community as an independent entity? The first question I will pursue in this thesis concerns investigating the nature of this organizational model.

1. What is the (theoretical) distinction between the sponsor firm and the sponsored community?

Herein, we need to find theories that are able to describe and pin-point the characteristics of Novell in relation to the openSUSE community. For this task I have selected Niklas Luhmann's theory of *autopoietic social systems*. This interesting theory has some peculiar aspects that make it particularly suited for our case. First, a counter-intuitive aspect of autopoietic social systems is that they do not consist of humans, but only their *communications*. Furthermore, the meaning of autopoietic is that the social system is continuously and *recursively recreated* by communicative events based on previous communications. Lastly, the sub-class of autopoietic systems that Luhmann terms *organizational* systems are distinguished from other social systems in that they consist of a special form of communication: *decisions*. By contrast Luhmann defines an *interactive* system as a more basic social network of communications, such as a board meeting or perhaps an academic discourse progressing over several years. In this thesis I will argue that these characteristics are able to distinguish Novell as an organizational system from the interactive openSUSE system, and that both systems are simultaneously evolving and altering their own boundaries towards each other.

With Luhmann's theory, we will be able to see how Novell and the openSUSE community are *separated* apart from each other. The relationship between these two systems are affected by the tension between openness and control, described above. The next step we need to take is to go deeper into investigating the exact nature of this relationship, and in particular how they are *held together* in collaboration.

2. *What is the nature of the relationship between the organizational and the external interactive system, and how are the two systems bound together?*

Given the differences in the commercial tradition of software development and the open source software model, multiple inherent conflicts could be present in Novell's open source adventure. If we also take into account the mentioned tension between openness and control in Novell's development model, many obstacles and difficulties complicate this collaborative attempt. I am therefore interested in seeking out the mechanisms, forces and arrangements that hold things together. In the thesis, I argue there are three elements that ensure a tight coupling between the two systems. By drawing upon the work of Susan Leigh Star (1989), I first discuss how several development tools and models serve as important *boundary objects* between the two systems, enabling joint development on a common product. Secondly, *shared communication channels* are vital in creating transparency and providing access through the boundaries of the systems. Thirdly, the *marginal people* whom have roles in both systems are crucial for balancing the needs of both of them. Based on this knowledge, I will turn to discuss some future scenarios for Novell and the openSUSE project towards the end of the discussion in this thesis.

The discussion of the case of Novell is as much an attempt to explore and test theories as it is an effort to describe and explain this empirical reality. Similar to much off-the-shelf software, social theories rarely solve the problem without adjusting, modifying and expanding them. In this study, this has been the third of my driving questions.

3. *How may the case of Novell strengthen our understanding of the theory of autopoietic systems and the theory of boundary objects, separately and in combination?*

In this thesis I argue that Luhmann's theory might fall short of explaining how social systems are linked together, and that the theory of boundary objects may be particularly suited at demonstrating system connectivity. Similarly, the case of Novell shows that Star's boundary objects are not only able to connect individuals and social groups, but also social *systems*.

A primary contribution to the theory of boundary objects is a distinction between what I describe as *supportive-objects* and *target-objects*, where the former serve as *means* and the latter as *ends* of collaboration (the target of the cooperation). Target-objects, in particular, have some very important characteristics that are crucial in explaining how heterogeneous actors are aligned in the development of a common product, and addresses an aspect I find lacking in Star's theory. The missing element to her theory is something that may explain the overall unity that binds actors together in collaboration, at a higher level than the pure mechanics of negotiating conflicts and aligning interests. By drawing upon Emile Durkheim's classic sociology, I try to show that target-objects such as the openSUSE object can serve as symbols and are charged with emotional

energy that unite the actors even when they are apart. I argue that despite their differences, the collaborative groups in fact inhabit the same social world in a form of *organic solidarity*. This is important for explaining how Star's boundary objects can hold common meaning and bind the separated groups together in the first place. Moreover, target-objects also contain some of the properties of epistemic objects (Knorr Cetina, 1997, 2001; Rheinberger, 1997), in that they hold an inherent motivating quality since they represent the end-goal and result of the collaboration. The collaborators are all stuck to the object because they *want* and desire to participate in its evolution. I argue that the motivating qualities of the object itself is largely neglected in the open source literature's explanation of individuals' motivation to participate on open source projects. This understanding should therefore be appended to this body of research.

Structure of the thesis

The structure of the thesis follows a somewhat different order than the one shown above. It is always a challenging pedagogical task to present a story where theory and empirical reality are entangled together. In this thesis they will nevertheless be discussed separately before they are merged towards the end. First, I will present the theoretical foundation for this thesis in chapter 2, where we will look at the research on open source firms and communities, Star's theory on boundary objects and Niklas Luhmann's autopoietic social systems. Thereafter I will discuss the methodology I have used to gather data for this thesis in chapter 3. In chapter 4, we will wander into the world of Novell and openSUSE where I will present my empirical findings, before we start the analysis and discussion of this reality in combination with our theories in chapter 5. In the final chapter I will outline some of the main issues in this thesis and discuss where further research may go.

Let's start!

Chapter 2

Theory

New organizational forms are emerging. The open source domain offers a window into new and interesting forms of work and organization, where formal membership is not required and contributors do not even receive a paycheck. What was for a long time perceived as hobbyist's activities has matured into an industry with a large scale of commercial interest. The unique form of organization of open source software development distinguishes it from most other familiar forms of production. When companies such as Novell move into the open source domain, the traditions, norms and forms of organization that exist in this social space can not be ignored. In the first part of this chapter I will therefore review the research on open source software development, and look closer at its distinctions in comparison to traditional commercial software development. It will also address the motivation for commercial software companies to pursue open source development, and some of the relations that emerge in the collaboration between companies and open source communities. Novell's initiative with the openSUSE project fits into a development model that is increasingly emerging in the software market, namely that of *firm-sponsored open source communities* (O'Mahony, 2007a; J. West & O'Mahony, 2005). The research on this area is however in its infancy (ibid), and I therefore seek to take our knowledge on the subject somewhat further in this thesis.

To shed some light into the dynamics of this organizational frontier, I have selected two theories. The notion of *boundary objects* (Star & Griesmer, 1989) as mediators of collaboration may help us understand how Novell and other firms are able to maintain a collaborative relationship to an

external community despite its conflictual nature. Second, I will use the theory of *autopoietic social systems* by Niklas Luhmann to discuss the boundaries between Novell and the openSUSE community, to help us understand what kind of organizational phenomenon we are dealing with. This pair of theories come from two separate trains of thought. In this thesis I hope to show how they may complement each other, and thus bind these two positions together.

The Open Source Domain

With the rise of the open source software movement there are now two models that account for the production in the software market: The *open source software model* and the more familiar and widespread *commercial software model*. Recognizing that open source software can also be used for commercial purposes, I nevertheless choose this phrasing as it captures the core motivation of software that is developed within this model, whereas the primary motivation of the open source software development model is to keep the software open and available for everyone to use - to whatever ends. The characteristics of the two models are summarized in table 1 below and detailed in the following section.

Characteristic	Commercial software model	Open source software model
Availability of code	Proprietary, closed source, restricted	Open source – free to use, modify, redistribute
Governance structure	Hierarchy (Williamson, 1987)	Voluntary participation and voluntary selection of tasks (Weber, 2004) Bazaar governance (Demil & Lecocq, 2006)
Rationale	Economic	Norm-based (Raymond, 2001)
Individual rewards of participation	Extrinsic: Monetary, salary	Intrinsic: Bugfixing, Learning (Hippel & Krogh, 2003), Career concerns (Lerner & Tirole, 2002)
Protection	Legislative: Copyright, patents, Secrecy (Dahlander, 2004)	Legislative: Copyleft Supportive foundations, Trademarking (O'Mahony, 2003)
Product examples	Microsoft Windows Adobe Photoshop	OpenOffice Apache Web Server

Table 1. Characteristics of the commercial and open source software model.

Models of software production

We typically find the commercial software model among firms in the commercial software market. As a development model, it is mainly characterized by a closed software development process where software code remains within the company. An important source of income

includes selling licenses for *use* of the software, which typically restricts the user from redistributing (copying) it to others. A company that develops a software product under the commercial model will strive to gain market shares at the expense of its competitors. As in most other business sectors, the competitive nature of markets encourages firms to be secretive about the product development, so that others may not copy or 'steal' their ideas or innovations. In the software business this means keeping the program source code unavailable to competitors, and patenting and 'copyrighting' product innovations. The obvious motivation of individuals working under the commercial model is mainly monetary (salary), although other extrinsic or intrinsic motivations certainly exist. The individual contributions are coordinated and governed within the structure of a hierarchy/firm (Williamson, 1987), and commercial software companies are typical carriers of this model. The model accounts for most software products that are sold on the software market today, including well known examples such as Microsoft Office or Norton Antivirus.

As the name suggests, the logic of the open source software model is distinctly opposite to the proprietary qualities of the commercial model. Software source code is open for all to see, use, modify and even redistribute. The end goal is not to make profits (although this also may happen), but rather to enforce the freedom of others to use the program source code as they please (Weber, 2004, p. 48). The availability of the code is secured through various licensing schemes, most notably through a legal mechanism that is referred to as 'copyleft' (Hippel & Krogh, 2003). This is a way of using copyright law to deny actors from taking the open software and later releasing it under a proprietary license that denies others access to the source code. In addition to licensing terms such as these, O'Mahony (2003) shows that open source communities use several mechanisms to protect the open source software from being 'eaten up' by commercial actors. These strategies include establishing foundations that sanction license violations and trademarking brands and logos. Although competition in the open source model does exist, for example between peers or communities who race to make the best solution to a given software problem, it does not lead to isolated software development in parallel organizations. Instead, the software source code is placed in reservoirs and commons where it is available for all programmers who wish to access it. In this way developers can build on each others contributions of source code. Through the Internet, there is basically no limit to how many programmers can contribute, while software development in a firm will be limited to the economic resources available and the number of engineers employed in the company.

Much of the literature on open source software development has been concerned with explaining why individual developers voluntarily contribute with loads of work for no pay (Morner, 2003). This research shows that rewards in the open source model are rarely economic, but rather consist of peer recognition and satisfaction in learning and accomplishing new tasks and problems

(Hippel & Krogh, 2003; Lerner & Tirole, 2002; Weber, 2004). This form of motivation is often described as intrinsic (coming from within), as opposed to extrinsic factors that are created by some external situation (such as being paid to do the work). Although the communal structure of open source may seem altruistic, it does not discount the fact that many motivational aspects are still egoistic. In sum, there is a large range of factors that may explain an individual's motivation to participate in an open source project. In figure 2 below, I have summarized the motivational characteristics based on how I find them categorized in the research literature.

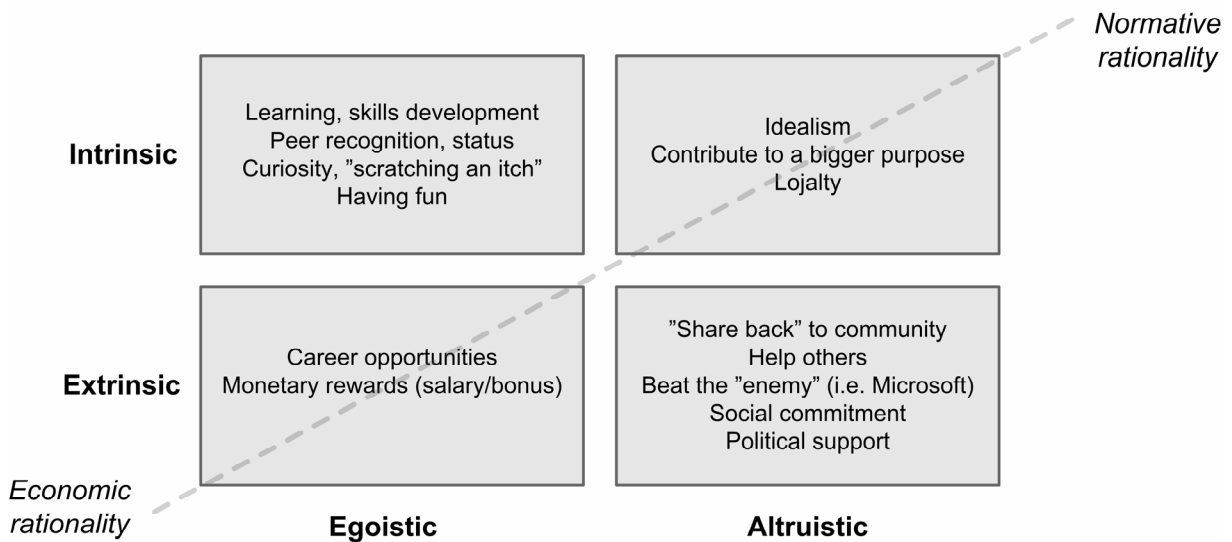


Figure 2. Motivational aspects of open source development

Within the world of software development, most software projects are associated with one of the two models of software production. We can view organizations and firms committed to commercial software development as structural carriers of the commercial software model. Similarly, we can say that the communities of open source developers (as a collection of individuals, groups and organizations) and firms involved with open source development reproduce the open source software model. I here understand *structure* in the sociological sense of Anthony Giddens (1984), as holding virtual existence only enacted through *practice*. The commercial or open source model only has a real-world existence through the actions of a software company or community. Furthermore, each of these models are *enabling*; they contain rules and recourses and serve as *normative structures* that enable actors in the software market to behave in a certain way. In this regard they also represent a form of rationality; within the commercial software model sharing source code is irrational, in the open source model it is the opposite. This dichotomy is useful for providing some insights to our first research question: we

could say that Novell and the openSUSE community, respectively, originate in and draw upon each of these two models of software production.

Reality is however not always as black-and-white as these models may indicate, and the organizations in the software world may not be so easily grouped in one or the other category above. An increasing amount of firms are becoming carriers of elements in both models, such as Novell and the openSUSE community do in combination. Other examples include Hewlett Packard, IBM, SUN and several others major IT-companies whom have chosen similar strategies. Giddens' structuration theory not only explains how structures reproduce themselves, but also how they are creatively produced or altered. His theory may therefore explain the emergence of a third structure in software production: the *hybrid software model*. Drawing upon the other two models, the hybrid model represents such a remaking of the traditional models of software production. After a brief short review of the history of open source, I will return to discuss how the open source and commercial software model may co-exist in the hybrid software model.

A brief history of open source

Where does open source software come from and when did it start? Although the main media attention surrounding the open source phenomenon has only been present for the last couple of decades, its origins trace back to the early stages of software development in the 1960's. Perhaps the most important software project at the time was the development of the UNIX operating system, originally created by Ken Thompson at AT&T's Bell laboratories. The development of UNIX found place in many different locations, and the distributions of UNIX at this time went practically free of charge without any effort to delineate property rights or to restrict use of the software that was made (Lerner & Tirole, 2002). Seeing that UNIX became immensely popular, AT&T realized that UNIX could be a commercial product. In the early 1980's AT&T began enforcing intellectual property rights related to UNIX. At Michigan Institute of Technology (MIT) a talented programmer named Richard Stallman was frustrated by the proprietary development of UNIX. Adding insult to injury, MIT had started licensing software written by researchers at the university to a commercial firm, restraining the open programming culture that had existed at the university. As a reaction to this development, Stallman established the GNU project (Gnu's Not UNIX) in 1983, a with the aim of developing a UNIX-clone operating system that would be free software without the use of UNIX code. Stallman also created the Free Software Foundation that would maintain the rights of free software projects. The main legal tool for achieving this is the GNU General Public License (GPL), which uses copyright law to secure that software and source code produced under this license is open for everyone. Anyone my use, edit and redistribute the free software as they see fit, but if you try to re-license it as proprietary software you are breaching the copyright of the authors. This legal maneuver was coined "copyleft" by Stallman, as it puts a twist to traditional copyright practices. In addition to ensuring

that the free software code stays open, the GPL also demands that derivative works from the free software remain free. This means that “it is not permitted under the GPL to combine a free program with a nonfree program unless the entire combination is then released as free software under the GPL”(Weber, 2004, p. 48). This mechanism is referred to as the “viral clause” of the GPL. With the General Public License and the Free Software Foundation to sanction any business that breaches this licence, some of the legal foundations for keeping open source open were in place.

At the end of the 1980’s, open source software was still not familiar to most computer users. Most personal computers ran the Microsoft DOS operating system with proprietary software applications. It was not until the mid 90’s, with the dawn of the Internet, that open source software development escalated. It was especially the development of the Linux operating system that put open source in the spotlight. As a graduate student at University of Helsinki, Linus Torvalds developed a crude and simple version of a UNIX-like operating system that he named Linux. In autumn 1991 he published the source code of a very early version of this system on an Internet newsgroup, asking for input and improvements from others. The response was overwhelming, and during the following years hundreds of developers were contributing code and new releases of the software was published on a daily basis. At the end of the decade Linux had become a major technological and market phenomenon, and by the mid-2000 Linux ran more than a third of the servers that make up the web (Weber, 2004).

Linux was licensed under the GPL, but not all software developers were happy with the strict terms in the General Public License. The term “free software” was also creating problems for business people (Hippel & Krogh, 2003). Even if the term was referring to free as in freedom (of expression), explaining this to the business world proved to be a hard pedagogical challenge. For this reason Eric Raymond and Bruce Perens coined the term “open source” and founded the Open Source Initiative (OSI) in 1998. The main task of the foundation was to create a definition of open source that could accommodate the GPL philosophy as a core, but which at the same time did not deny proprietary affiliations to the open source software product. The official open source definition created by OSI neither contains the viral clause found in the GPL, nor does it place restrictions on other software that is distributed along with the licensed software⁴. Netscape was the first major commercial company to put this new definition to use, when they released the source code for their browser Netscape Navigator (later to become Mozilla) in 1998.

⁴ From the open source definition at www.opensource.org.

Commercializing open source, open-sourcing commerce

How does a firm make money with open source software? If we use the terms I have introduced so far, the question can be phrased in another and perhaps more interesting way: How can the open source and the commercial software model work together? This question includes a political and a social dimension, in addition to the economic perspective in the former phrasing. The two software models are somewhat contradictory in nature, which raises several interesting issues. I will start by discussing the economic foundations of open source, by presenting various constellations of the hybrid software model. These are often called open source business models in the literature I will be reviewing. Later I will tend to the different kinds of social relationships we may find between firms and open source communities.

It is important to clarify one common misunderstanding: Free software (a name used interchangeably with open source software) does not mean software being free of cost (*gratis*). While open source software in very many cases *is* free of cost, this is not a requirement in any of the open source definitions or template licenses of open source software. “Free” in this sense refers to the *freedom* to use, modify, improve, redistribute or otherwise do you want with the software, but you still might have to pay for the free software. (The ironic sound to this last sentence is one of the reasons Raymond and others coined the alternative term “open source”). That being said, it is still hard to understand how one could make money on a public good available to everyone. Even if it is perfectly legal to charge a fee for an open source product, how much sense does it make to pay lots of money for a product you can download and install yourself free of charge? There has been done some research on the issue of appropriating returns with open source software (Bonaccorsi et al., 2007; Dahlander, 2004; Lerner & Tirole, 2002). The findings identify multiple sub-strategies to selling open source products and services. I will describe these various strategies in the following paragraphs. The strategies are summarized in table 2 (the table builds on a similar table provided by Dahlander, 2004). Note that an open source firm rarely pursues a single strategy, but rather combines several of them.

Products:	Packaging	Facilitating and distributing open source products for easy use.
	Proprietizing	Adding proprietary solutions to an open source product and selling licenses for the new product.
	Spin-off	Selling complementary proprietary software products surrounding a major open source project.
	Black-box	Bundling and integrating pieces of open source software products in a hardware solution.
Services:	Education and training	Education based on an area of expertise within open source software.
	Consultancy	Consultancy work based on an area of expertise within open source software.
	Support	Support based on an area of expertise within open source software.

Table 2. Business models with open source software. Modified from Dahlander (2004).

First of all, it is possible to sell an existing open source software package in its given form, and charge a fee for preparing and distributing the software. Not all open source software programs are easily accessible, and sometimes it is of value to have someone prepare the software for use. For example, a small company may not have computer engineers available to access, compile and configure the source code and install the software. They would therefore be willing to pay a third party for delivering the software in an easy-to-install package from the open source community. There are limits as to how much value can be added in this manor, as demand will decrease as the price exceeds the cost of hiring someone competent yourself to do the job for you. Nevertheless, the task of bundling together multiple applications with a compatible operating system can be quite substantial, which creates business opportunities for companies that specialize in such work. I term this product sub strategy as ‘packaging’. Second, a common product strategy is to add proprietary modules (an additional part to the program that is not open source) to an existing software package, or make proprietary improvements to the program and sell licences for the new product. This is only possible if the original software is not licensed under the GPL or other licencing scheme containing a viral clause that prohibits such ‘proprietizing’ of the software. Mixing open source with closed source software is a common strategy for many open source companies. A third product strategy is to sell proprietary spin-off products to an established open source product. This could for example be proprietary software plug-ins for a large open source application, or server management tools for Linux servers. In this case the firm is selling a stand-alone proprietary product, and as such might not fit the definition of an open source firm. But chances are that the firm will be interested in the quality and use of the open source product, and would thus be contributing to the software or otherwise participating in the community

surrounding that product. A possible way of creating a market for the spin-off products could be for a company to release the source code for an established proprietary product, while selling licenses for complementary software. If the main software originates from the same company, it will benefit from having the highest level of insight in that technology, and may thus create the best complementary products. By releasing an established proprietary product as open source, a company may also hope to increase the user-base of that product to a level that makes it an industry standard. This can then provide the company with additional long-term benefits (for example building a reputation and a trademark that provide users with trust in future products). A fourth strategy consists of integrating open source software with a hardware solution, and selling the resulting integrated product. This is called “black-boxing”. SUN Microsystems started out by selling hardware solutions configured with open source-UNIX, and have been successful with the black-boxing strategy.

The services-strategies are pretty straightforward, and differ little from the similar services related to proprietary software products. Dahlander defines three types of services: consultancy, support and education. In a case study of six Nordic open source firms, he found that all six companies were involved with consultancy work, while half the number provided education services (training in the use of the system/software provided by the firm).

The strategies outlined above represent some of the most common ways of appropriating returns with open source software. It would be a mistake to assume that this is a complete list of all possible strategies. New and creative strategies are constantly emerging. In the case of MySQL, for example, this Swedish firm has achieved success with an interesting dual licencing scheme (Dahlander & Magnusson, 2005). While the MySQL database software is licenced under the GPL (containing the viral clause that prohibits ‘proprietyization’ of the software), it is also possible to buy an alternative licence of the software that allows the user to keep any proprietary additions and improvements to the software the user develops. In other words, if you want to keep your innovations secret from the community, you have to pay for it. As mentioned, a software company may base its strategy on various constellations of the strategies presented above. Novell, for example, packages a Linux distribution, combines it with proprietary products that run on top of it, and offers all types of services for it.

Community and firm relations

The strategies outlined above are examples of interaction between carriers of the commercial and the open source software model. The initiative to pursue a hybrid model may come from both sides; open source communities may start up businesses to profit on their software projects and at some level seek commercial solutions, or commercial software companies may turn to open source communities to find new resources. It is interesting to ask how the open source

communities react to the presence of large companies that profit on the communities efforts. This will most likely depend on the nature of the relationship. Bonaccorsi et al. (2007) are currently researching the question of whether for-profit firms act not only as *takers* but also as *givers* by contributing to open source projects. A similar dichotomy can be found in Dahlander's early work on the case study of six Nordic OSS-firms:

An interesting finding was that the cases outlined two main types of firms - 1) "FRIENDS" who are active in getting their product widely diffused through releasing code and establishing communities, and 2) "FOES" who only attempt to capitalize on the work on developers and users in the community. FRIENDS are more aligned with the community by giving away code and establishing projects, whereas FOES only appropriate the joint effort of the OSS community by acquiring knowledge in a given area and sell their expertise to customers. (...) The rationale for FOES is to minimize R&D investments and time to market through using existing OSS modules (Dahlander, 2004, p. 18).

With this terminology it is clear that the firms' commitment and strategy towards the community will greatly impact the social relations and sentiments between them. Understanding and uncovering the nature of these relations is important for discussing the sustainability of the integration between the commercial and open source model. Dahlander and Magnusson elaborate on these relationships. Based on case studies they find that relationships between firms and open source communities may be characterized as *symbiotic* or *parasitic* (Dahlander & Magnusson, 2005). The symbiotic approach implies an interdependent co-evolution between the community and the firm. Legitimacy is gained through status in the community based on its norms and values, not from having a formal role in a firm. In the parasitic approach it is clearer that the firms maximize their own benefits often in violation with the norms of the community. A parasitic relationship could for example characterize a firm with a packaging or proprietizing strategy, making large profits as a free-rider without contributing anything back to the community. The researchers also find examples of relationships that are somewhere in-between these two.

For companies that seek to establish a sustainable relationship with open source communities, it is clear that they will want to be perceived and act in a symbiotic fashion. To strengthen such a perception it would be important to deliver quality contributions, at least as an act of good will. This will be more important for large companies than smaller firms, as the participation of the latter might go unnoticed. For companies that seek to establish their own open source projects and gather voluntary developers, it is particularly important to create a symbiotic identity among open source communities as it is important to be *trusted*. In open source communities trust can be institutionalized in various ways. Novell, for example, uses the GPL license for all of their open source products. This is likely to be very important in establishing faith and trust in the

company as a legitimate open source actor, having a background as a large proprietary incumbent.

Willingness and resistance

The discussion has so far addressed the distinctions of open source and how this model can relate to commercial practice. We will now have a look at some of the underlying motivations for pursuing and resisting such integration – from both sides. The values, norms and motivation factors described below are embedded in the normative structure of the open source and the proprietary software model, and contribute in forming the rationality that guides the actions of actors in the social space of software production.

From the commercial point of view, some arguments for why open source is interesting have so far been presented. Open source is as a resource that can be of commercial interest for many reasons, in addition to the monetary strategies detailed in the previous chapter. For one, companies can use existing open source software as part of their commercial products, and save efforts from having to develop it singularly. Second, by making their own products into open source projects companies can benefit from cooperating with a large group of talented developers outside the company. The contributions from the community can then be used for further product development. In other words the company may gain access to a large amount of human capital - free of cost. Furthermore, the ‘open source’ brand itself may create positive sentiment among potential customers, creating a sense of legitimacy or trustworthiness. When open source is used a resource in this manor, it may in turn contribute to the end goal of creating profits for the company. Note, however, that the sentiment may also be directed the opposite way among customers with negative feelings towards open source software. It is interesting to observe how the perception of open source has evolved over the last years. Microsoft has, for example, moved from claiming that open source licenses are a ‘cancer’⁵ in 2001 to launching their own open source initiatives during the last year⁶.

Are there any benefits for the open source community in this partnership, or is open source only exploited as a resource? There seems to be several rational gains for the open source community as well. For one, it is of interest to the open source community to receive software contributions from the commercial companies (or any actors for that matter). Software companies and corporations possess a large amount of recourses. In the example of Novell, detailed later, the company has 800 engineers employed that work exclusively on open source software that is shared with the larger community. Second, it is a triumph if a given commercial software project is released as open source, rather than being enclosed as a proprietary product: it becomes a

⁵ See http://www.theregister.co.uk/2001/06/02/ballmer_linux_is_a_cancer/

⁶ See <http://www.microsoft.com/opensource/default.aspx>

public good. Third, integration with the commercial model contributes to reproducing open source software model. If commercial activity and open source were completely incompatible, the number of people contributing to open source development would be substantially smaller, and maybe too small to make open source survive as something more than just an idea practiced by few. This rationale fits well with the pragmatic BSD-style license and the ideology of the Open Source initiative. The characteristic argument from this perspective is that as long as the integration is creating more open source users and products, it is a positive thing. This practical approach is clearly expressed in the official open source definition:

“The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research. Rationale: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it” (§6, Open Source Definition).

There are also arguments against and forces that resist such integration. From an economic perspective, open source will not be an option if it is not considered or proven to be profitable within an expected time-frame. Second, open source uses a methodology that may be considered unreliable or untrustworthy from a certain point of view. Since open source is based on voluntary participation and voluntary selection of tasks, there is a risk that certain areas of work does not attract developers and lay barren. Whereas a commercial company can assign tasks (also boring ones) to developers with the use of formal authority, an open source community does not have the same administrative fiat or control to ‘force’ someone to do the job. Although most communities would certainly argue that there are alternative mechanisms in place to avoid this problem, it may nevertheless create uncertainty from the viewpoint of a firm.

Although a substantial part of the open source movement is motivated for commercial integration, there are also those who oppose. The ideological principles of the Free Software movement, for example, are not sympathetic to blending open source software with the proprietary model. With the title “Why Software Should Not Have Owners” in one of his essays, Stallman clearly positions the Free Software movement in opposition to the conception of property rights in the commercial software industry (Stallman & Gay, 2002). Again, the foundation also opposes the term “open source” as this phrasing may “sell-out” the principles of free software for the purpose of appealing to business users. The Free Software movement has therefore insisted on being distinguished from the Open Source Initiative. Nevertheless, Stallman argues that the differences between the Open Source Initiative and his foundation are marginal compared to the differences toward the commercial model: “We disagree on the basic principles [with the Open Source movement], but agree more or less on the practical recommendations. (...) We don’t think of the Open Source movement as an enemy. The enemy is proprietary software”

(Stallman & Gay, 2002, p. 55). And indeed they have some common issues. Even the supporters of commercialization within the Open Source movement do not want the integration with the world of commerce to happen unconditionally. All actors within the movement are aware that open source is vulnerable to appropriation unless necessary precautions are taken. Therefore the open source communities have been able to establish various mechanisms for ‘guarding the commons’, which have been well documented by Siobhan O’Mahony (2003). These involve licensing schemes, establishing foundations that sanction license violations and trade-marking brands and logos.

Firm-sponsored open source communities

The interest for open source as an intellectual property strategy and as a development methodology has grown substantially among commercial software companies. Firms have in the recent years taken a step further than simply participating in open source communities and using open source software. Many previous proprietary software companies have chosen to open up their own software projects and attempted to create surrounding open source communities, and “spinout” their internally developed code (Joel West, 2003). Netscape’s launch of the Mozilla project presented the first case in 1998. Further examples include Apple and the Darwin project, IBM and Eclipse, SUN Microsystems and the OpenOffice project, and more recently Novell and the openSUSE project. While autonomous open source software communities have received a great deal of empirical and scholarly attention within the last decade, there has been very little research on corporate sponsored open source communities such as these. The exceptions are (O’Mahony, 2007a; Joel West & Gallagher, 2004; J. West & O’Mahony, 2005). In the following I will detail what we *do* know about sponsored communities, based on this research.

Why do companies release their source code and start communities? The previous discussion has shown how companies may monetize investments in open source products in general. According to West (2003), a specific reason for a firm to “open source” its products may be to *win adoption* or to *gain development assistance* on non-critical areas. O’Mahony and West argue together that the reasons for external users to adopt the technology or make contributions in such cases are first that companies have resources and put in investments that may assure a solid technical foundation of the innovation (2005). Secondly, the project has a clear owner that can secure the progress of the project; there is someone in charge. “On-going sponsorship provides recourse, legitimacy and technical capabilities to improve the odds of project success” (J. West & O’Mahony, 2005, p. 3).

There are also several challenges that are raised by such an initiative. Three issues identified by the same authors involve technical, relational and legal aspects: First, in an autonomous, community-managed project the code-base tends to develop and scale simultaneously with the size of the

community. In sponsored projects, however, the code base can be fully matured and quite complex at the time it is released. This can make it difficult for outsiders to contribute, as they have to relate to the technology *post hoc* of conception. Secondly, community members tend to lack any ownership in the project, as it may be fully controlled by the sponsor. This can result in a lack of intrinsic motivation often found among members of autonomous community projects. Finally, contributors might worry about the future legal rights of the code they develop, which becomes owned by the company. Although various licensing schemes (such as the GPL) may create confidence, neither of these licenses have in fact been tried out in court. Should a conflict about the code emerge, the contributor's interests are not guaranteed.

O'Mahony has investigated the differences between sponsored and autonomous communities further, by conducting a comparative study of a dozen well-known sponsored communities and compared them to autonomous communities (O'Mahony, 2007a; O'Mahony, 2007b). Her main finding is that sponsored communities have to manage a tension between *openness* and *control*. This characteristic is most certainly present in the relationship between Novell and the openSUSE community. While Novell wishes to attract external developers to participate in product development, they have great difficulties with granting them direct access to change the code of the main product as Novell must be able to guarantee the quality of the code to its many customers. We will return to the case and discuss it in closer detail later. O'Mahony makes a distinction is made between two types of openness: *transparency* (ability to read code, rights to use it and visible governance structures) and *accessibility* (ability to change code, ability to participate in governance and control of derivative code). While both types of communities offer a high degree of transparency, accessibility was dramatically different between sponsored and community managed projects. "Governance of autonomous projects was largely pluralistic, shared widely among community members, whereas the ultimate decisions of sponsored communities were (with rare exceptions) controlled by the sponsor" (O'Mahony, 2007a). Many sponsors were reluctant to give up control of the project since it would reduce their ability to align the community's production of open source software with their corporate strategy.

Another interesting finding is however this: while informants claimed that high levels of transparency aided the adoption of the products, there was no direct evidence of the direction of causality between participation and lack of accessibility. In the case of MySQL – where the code is tightly controlled by the founding company – the high rate of participation suggests that accessibility is only *one* of the factors that drive participation. Nevertheless, sponsors that valued outside contributions the most, recognized that they could only hope to attract the most talented programmers outside the firm by devolving some level of control (ibid, p.14). Another factor that may influence outside participation in the sponsored community, are the technical aspects of the project. Two factors can possibly help the growth of the community: Modular software

architecture allows the participants to easily pursue work on areas of their interest (and not having to deal with the rest). Secondly, a clean coding style that is well documented increases readability for external developers.

O'Mahony recognizes that further research is needed to test whether the constructs above can fully explain the fate and success of a sponsored community. Other factors such as product quality, size of target market, size of existing user-base or price of competitive products may also have an influential value. Also, more research is needed on sponsored communities' trajectory of development – how does the tension between openness and control evolve over time? And what are the detailed mechanisms of managing this tension? The next sections will introduce the theoretical framework that I will use to address answers to these and other questions.

Boundary objects

In the social space of software development there are several groups in interaction, competition or collaboration. They may represent different views and interests, with economic, egoistic or even altruistic motives. Physically, they can be represented as organizations, ranging from commercial firms to voluntary software projects. Novell and the openSUSE community are examples of such. The groups are composed of individual software programmers and even non-technical contributors, who often take part in several communities and organizations simultaneously. These organizations and communities within them may be described as various *communities of practice* (Wenger, 1998) where individuals start out as legitimate peripheral participants and become full members once they learn the ropes. Learning and the creation of meaning and identity go hand in hand, and fuel each other in the development of each individual and the community as a whole. In another perspective, we might characterize these groups as inhabitants of different social worlds (Strauss, 1978).

These social groups are confined within *boundaries* that separate what is inside (i.e. group members) to that which is outside (other groups and people, the environment). Boundaries between social groups may be understood in various ways. In communities of practice these are defined by the individual's *participation* and *practice* (Wenger, 1998). If you participate, you are part of the community. Other strands of sociology and anthropology focus on boundaries in terms of *identity* and *belongingness* (Barth, 1969; Lamont, 2001; Turner, 2001). A typical representation of boundaries is made clear when people speak of a separation between “us” and “them”. Identity may relate to static properties such as ethnicity and gender, to semi-changeable properties such as educational background and occupation, or fluent aspects such as political views or music you like to listen to. In traditional organizational theory boundaries are commonly constituted by formal membership (eg. through employment contracts). Some organizational

research has been concerned with studying collaboration between groups separated by *knowledge boundaries* (Brown & Duguid, 1991; Carlile, 2002). This is typical for interdisciplinary work, where group members occupy different domains of knowledge. What you do *not* know in relation to another group thus becomes a boundary. None of these definitions of boundaries necessarily contradict each other. We may often speak of all these types of boundaries surrounding one and the same social group. Although boundaries are often considered as lines that separate groups from one another, it is important to acknowledge that boundaries also *bind groups together*: they only emerge when two entities meet. This characteristic is often neglected in analytic work on social groups, but will play an important role in the theory that will be outlined below.

The topic for this thesis is to study how collaboration is made possible at the intersection of such boundaries, more specifically by studying the model of firm-sponsored open source communities as represented by Novell and the openSUSE project. In this model, two main groups are linked together in product development – a commercial software company and an open source community sponsored by the company. The product development may also be linked to a mass of other (open source) communities surrounding each component of the larger software product. The challenge is then to understand what keeps them together in cooperation, despite their differences in motivation, knowledge, norms and organizational identity? The notion of *boundary objects* (Bowker & Star, 1999; Carlile, 2002, 2004; Star & Griesmer, 1989; Wenger, 1998) has proved particularly well suited in dealing with issues of collaboration across such boundaries.

What is a boundary object?

The main question that the theory of boundary objects is meant to address is how collaboration is made possible among people that inhabit different social worlds (Star & Griesmer, 1989). As an example, Paul Carlile (2004) analyzed collaborative work between different departments within a automobile company. Each of these departments represents their own domain-specific knowledge about various aspects of the car product (vehicle styling, engine, safety and climate control). Traditionally, a clay model of the car had been the main boundary object at the intersection of all departments. This object was however not strong enough to represent the views of all groups (it favored the vehicle styling group and did not represent climate control-concerns very well), and was later replaced by a powerful vehicle simulation tool that was able to represent the interests of all departments, allowing them to reach reconciliation of meaning in the design of the car. In this case, the social worlds were separated by the knowledge they each represented. In another example, Susan Leigh Star's (1989) original research on boundary objects discusses collaboration between scientists, administrators, entrepreneurs and hobbyists (among others). They too represent different bodies of knowledge, but are in addition more distinctly separated as social

groups with diverging interests – they hold fewer properties in common than the employees in the automobile company.

When such groups come together to solve a common problem, the success of their collaboration can not be taken for granted. There may be several obstacles standing in between, such as knowledge boundaries, disagreement concerning the problem definition and different political interests. Star and Griesmer provide a theoretical framework for overcoming these obstacles, as boundary objects can leverage and mediate collaboration despite the presence of difficulties. The primary feature of boundary objects is that they exist in all of the intersecting domains. More specifically boundary objects are “both adaptable to different viewpoints and robust enough to maintain identity across them” (Star & Griesmer, 1989, p. 387). The theory has been applied to a large amount of empirical research, where examples of boundary objects range from repositories, sketches and drawings, workflow matrices, physical and IT objects, prototypes to more abstract objects such as metaphors, narratives or processes and methods (Nicolini, Mengis, & Swan, 2007). Boundary objects are often internally heterogeneous. They can be simultaneously concrete and abstract, specific and general, conventional and customized. “The nature of the problem determines what is adequate concreteness for a given boundary object” (Carlile, 2002, p. 452). They are flexible and can adapt to local needs in individual communities, while they can still maintain a common identity *across sites*.

We can summarize the distinction of boundary objects in two particularly important properties that make them so adept at dealing with cross-disciplinary collaboration: (1) they contain shared and common knowledge across groups and maintain the identity of all groups. They can “belong” to everyone, despite different interpretations of meaning between the groups. (2) Boundary objects are also able to handle different interests between collaborating groups, and may even facilitate negotiation between them. In the terminology of Latour and Callon (1986), the objects support *translations* by multiple actors. In the following I will present these properties in closer detail.

Shared knowledge, mutual identity

A contrast to a boundary object can be an object that is identifying of a community, and that may *not* be shared with other communities. Becoming a member and learning the ropes within a given community of practice may involve becoming familiar with such objects. In Wenger’s view, learning and membership within a community of practice are one and the same, as you move from being a *legitimate peripheral participant* to becoming a full member. This process entails a series of encounters with the objects involved in the practice (Bowker & Star, 1999).

Anthropologists refer to the process of creating the familiarity of categories or objects within a community as *naturalization*. “Illegitimacy, then, is seeing those objects as would a stranger [...]

Membership can thus be described as the experience of encountering objects and increasingly being in a naturalized relationship with them” (Bowker & Star, 1999, p. 295). Carlile (2002) refers to these as *within-practice objects*. In contrast, boundary objects may be understood as *across-practice objects*.

An important property of a boundary object is its ability to carry and contain knowledge that is shared between (or across) the groups in collaboration, in contrast to naturalized objects which only make sense within a group. This ability relates to the fact that the object has an identity in all groups: “(...) in conducting collective work, people coming together from different social worlds frequently have the experience of addressing an object that has a different meaning for each of them. Each social world has partial jurisdiction over the recourses represented by that object, and mismatches caused by the overlap become problems of negotiation” (Star & Griesmer, 1989, p. 412). In other words, boundary objects may have a different ‘faces’ in each world, but at the same time they maintain a common identity across sites. Consider, for example, the role of the patient record in the work of healing a patient at a hospital. Both doctors, nurses, pharmaceuticals, secretaries and patients (to name a few) are involved with the work of treating a patient. The patient record is an important object for all of them, as it contains the common knowledge of the patient treatment that everyone makes use of. Which aspects of this object and how the information is interpreted may however be different for each group. For a lab-technician, entries into the patient-record may mark the final stage of her work-process. For the doctor, the same event could be the start; the results indicating what medical treatment to pursue and medication to prescribe. For the patient, the lab-results may indicate how many days he will be ill, while the secretary will use the information to figure out how many nights she will have to make a bed available for the patient at the hospital.

The fact that boundary objects are able to maintain a common identity between groups actually addresses an issue I find problematic in Star’s theory. Star refers to the various groups surrounding these objects as inhabitants of *different social worlds*. If we investigate this term closer, it might however be difficult to agree upon such an understanding, as a common social world may be a precondition for an object to maintain a mutual identity between groups within it. For a social world to exist, it should at some level be an autonomous entity that is able to *support its own identity*. I therefore argue that the boundary of a social world can not be drawn around a department of a vehicle company, as in the earlier example. The climate control group would never exist without the larger automobile company. The mother company unites the groups within it, and the commonalities that all engineers share as employees of this company make them inhabit this social world *together*. The distinction of the groups and departments in the company is the result of a division of labor. In the words of Emile Durkheim, we might say the car company is unified by an ‘organic solidarity’. Durkheim uses this term in contrast to the

'mechanical solidarity' often found in traditional societies, where the lives of its members are homogeneous and people have similar experiences and conceptions in common (Hughes, Martin, & Sharrock, 2003). Modern society, however, is distinguished by its high division of labor and specialization between its members, resulting in a society of interdependent and mutual indispensable individuals. Although most citizens of Paris might not know each other, they are all part of the same society and rely on each others services and specialties. Societies such as these might not hold the strong, shared sentiments which are the basis for the unity and conformity found in mechanically solitary societies, but they are nevertheless united by a larger whole. Durkheim draws the term 'organic' as an analogy from biology, and "refers to the kind of unity found in the organism in which differentiated and specialized parts are combined into a single, functioning whole, with each part's own operations depending upon the whole" (Hughes et al., 2003, p. 165). Returning to our discussion, the distinction of separated social groups, practices and boundaries between them may without doubt be present *within* a social world. I however argue that the social world is *shared*, as in an organic solitary society. This understanding is important in explaining how, for instance, a clay model of a car may give meaning to all groups within the automobile company. On the basis of this argument, I would claim that Star's theory needs support for explaining how collaborators are united at a higher level than just as self-interest seeking groups. I will return to this discussion in chapter 5. Now we will have a closer look at the importance of objects for handling multiple interests in collaboration.

Translation, negotiation and reconciliation

When Star originally coined the term, the theory of boundary objects was developed to understand how various groups of actors managed to collaborate in order to establish and develop the Museum of Vertebrate Zoology at Berkeley successfully. Their initial problem was to figure out how the scientists, entrepreneurs, university administration, hobbyists and animal-trappers - with diverging interests and different stakes in this work - managed to reconcile their differences and collaborate. They started off by drawing upon Michel Callon's concept of *translation* (Callon, 1986), which addresses the issue of power and the process of imposing one's interest onto others so that it becomes their own (described closer below). However, Callon does not explain what happens when a set of actors with different interests try to impose these simultaneously, or perform what Star and Griesmer call "n-way translations". Creating a solution to this problem was part of their mission, which resulted in the "discovery" of boundary objects. Boundary objects are inherently conflictual, as they convey different interpretations from several groups. They however manage to balance these conflicts, and support multiple translations. This aspect is however sometimes ignored in empirical research that utilizes this theoretical framework, such as in Nicolini et.al (2007) where the emphasis is put on boundary objects' brilliant ability of "coordinating the work" of interdisciplinary groups. If we reduce the concept to such a simplified

and convenient understanding, we stand the risk of treating boundary objects as the “magic bullet” that Paul Carlile warns us from doing (Carlile, 2002, 2004). To understand this process of translation, it may be helpful to visit Callon’s original use of this theoretical construct.

The term “translation” may itself be misleading, which might be the reason that Callon’s perspective sometimes falls out of a typical description of boundary objects. Our common understanding of the word is related to language, and involves transforming someone’s representation of meaning into a more accessible form. This definition of translation is descriptive by nature. Callon however scales this definition to include the transformation of *interests*, and thus adds a normative side to the word. Translation becomes a process where an actor tries to translate one’s interest into that of another (sort of making your goals become theirs). “To translate is to express in one’s own language what others say and want, why they act in the way they do and how they associate with each other: it is to establish oneself as a spokesman” (Callon, 1986, p. 19). A translation then becomes a political process of power. A typical definition of power is understood as the ability to make someone do something they otherwise would not. This is also the purpose of Callon’s translation, although this theory emphasizes that the other party will yet have his own goals fulfilled. It is therefore a process of aligning interests.

The theory of translation is associated with actor-network theory (ANT) (Law & Hassard, 1999), a framework developed by Bruno Latour originally for studying science and technology. In ANT, an actor does not need to be a human being (for which reason the term “actant” has later been adopted by Latour to avoid confusion (Lee, Liebenau, & DeGross, 1997)). The theory of translation is thus also applicable to both technical and non-technical actors. In fact, Callon presented translation theory to support his general argument that the same theoretical framework should be applied to both domains of society and nature, which have always been theoretically separated. Translation theory is an example of such a theory. To understand how the process of translation works, Callon created four supporting constructs that describe different “stages” of the translation process. The first stage is *problematization*, where an actor starts with creating the problem definition that others must agree upon. In order to reach their goals, other actors must pursue a course defined by the first actor (see figure 3). This course is called an *obligatory point of passage* (Callon, 1986). By establishing the obligatory passage point the actor may maneuver himself to become indispensable in the network. Creating an obligatory passage point is to identify what the various actors “want” on their behalf, and make them form an alliance around it. An example might be helpful: The United Nations Climate panel has been largely successful in establishing the target of reducing CO² emissions as the obligatory passage point in reducing global warming. This is the course all nation-states should pursue and what the industry must

accept. The climate panel has also become the undisputed authority on the issue, and have been awarded the Nobel peace prize for their work (and rightly so, in my opinion).

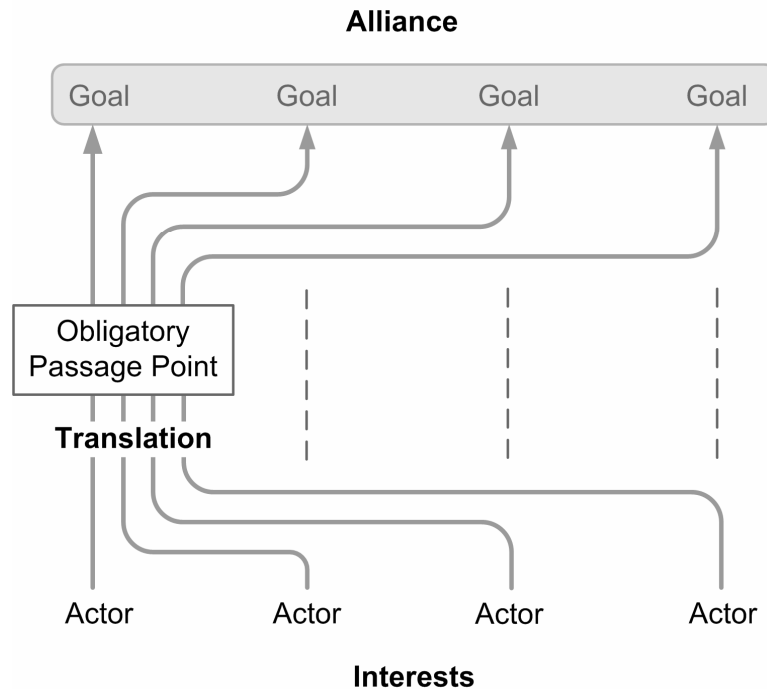


Figure 3. Translation. Modified from Callon (1986).

Problematization also includes defining the actors themselves. In the next stage of *interressement*, this definition is cemented. “Interressement is the group of actions by which an entity attempts to impose and stabilize the identity of the other actors it defines through its problematization” (Callon, 1986, p. 8). This involves the use of *interressement devices* that cut off any disturbing elements that may be pulling the actors towards another definition. Callon also refers to this process as locking the allies into place. It is nevertheless at the third stage of *enrolment* that this interressement is put to the test, where negotiations and trials of strength will show if the interressement succeeds. Enrolment occurs when the actors comply with the problematization through behavior.

The final stage in the translation process concerns the *mobilization* of allies. If the main actor has been successful with all the previous stages, he may become a spokesperson for all the other actors and speak on their behalf. This means that the actor can mobilize these actors (by speaking on their behalf) in contexts in which they are not present.

In sum, “[t]ranslation is the mechanism by which the social and natural worlds progressively take form. The result is a situation in which certain entities control others” (Callon, 1986, p. 19). This

theory may be very helpful in understanding the political aspects of interdisciplinary collaboration, but also has an important weakness. Translation theory explains how one actor, A, goes about with imposing his interests through the stages of problematization, interesement, enrolment and mobilization. In this story, actors B, C, D, etc are left to be dealt with by actor A, and to respond to these actions (becoming enrolled if the translation is successful). However, what happens when B, C, D have their own agendas, and initiate their own conflicting processes of translation? It does not seem as if the theory of translation is able to grasp the complexity of multiple actors imposing their conflicting interests simultaneously, but that is where Star and Griesmer's *n-way translations* make a difference. Interdisciplinary or boundary-spanning work is not driven by consensus, in fact, "consensus is not necessary for cooperation nor for the successful conduct of work" (Star & Griesmer, 1989, p. 388). Such collaboration is therefore composed of an internal diversity of interests and meanings, which somehow need to be reconciled if the collaboration is to succeed. Figure 4 shows how boundary objects may serve to support coherence in a many-to-many mapping of interests and translations, where there may be several passage points that can be maintained at the same time.

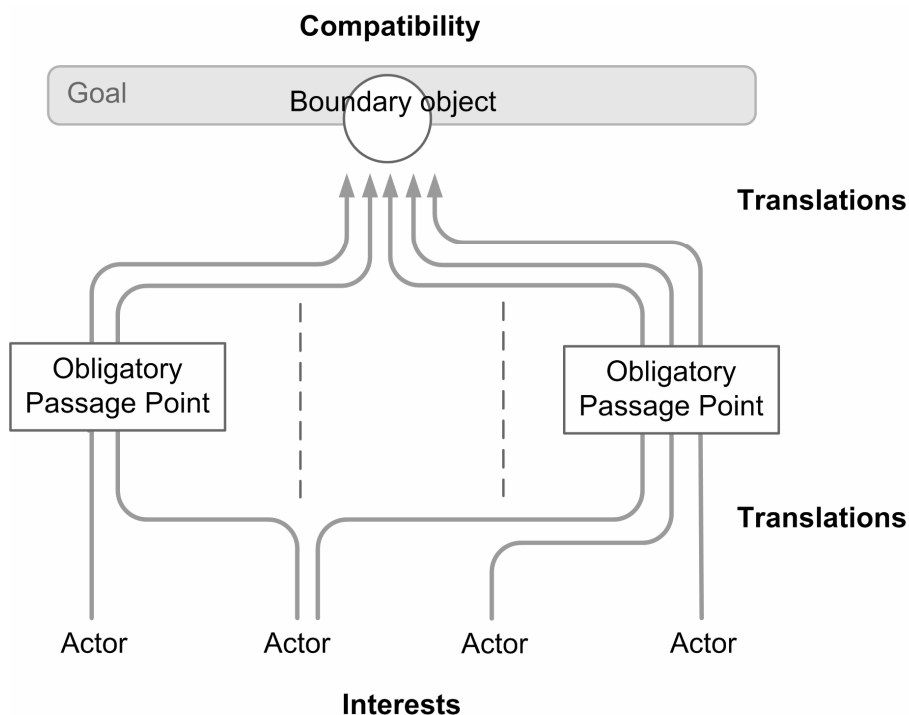


Figure 4. Boundary objects and translations. Modified from Star and Griesmer (1989)

An important feature of translation is that it is not about simple coercion. Each translator must maintain the integrity of the interests of the other audiences in order to retain them as allies.

Collaboration between different groups of actors therefore requires reconciliation of meaning between their respective social worlds. Another way of referring to this process is to say that boundary objects are able to *make different interests compatible* with each other. This is represented at the top in the model above. In the collaboration between Novell and the openSUSE project, various interests are represented through the normative structure of the open source and proprietary software model. Advocates of these models promote different passage points, which are made compatible in ways that will be discussed later.

Types of objects

When multiple interests, knowledge areas and identities come together in objects at the intersection of social groups that are joined in the solving of a problem, collaboration between these groups is made possible and facilitated by these same objects. However, boundary objects are not magic bullets: they are created through practice and only maintained through continuous practice. What is a boundary object in one context may not be one in another (Carlile, 2002). Sometimes the relative nature of these objects can make them hard to identify. Star's theory does not set any clear limits as to what can and can not be boundary objects, as long as they contain the properties described above. Therefore, the understanding of boundary objects has the ability to scale from small, tangible objects within communities of work (such as workflow sheets) to large and abstract objects, such as the understanding of "labor law" in negotiations between unions and employers.

Boundary objects may also be divided into several *types*. Star originally created four categories of boundary objects: repositories, ideal types, coincident boundaries and standardized forms (Star & Griesmer, 1989). This list is not exhaustive, and only represents an analytic distinction between the objects. Two of them are worth a closer presentation, as we will find them present at Novell's boundaries of development later on. *Repositories* are ordered 'piles' of objects which are indexed in a standardized fashion. An example of a repository could be a physical library or an electronic database. Repositories have the advantage of modularity, as "people from different worlds can use or borrow from the pile for their own purposes without having directly to negotiate differences in purpose" (Star & Griesmer, 1989, p. 410). *Ideal types* is the other significant category for the case of Novell, and represent objects that hold *symbolic value* for the collaborative participants. This could typically be something like a diagram, model, map or atlas. It is abstracted from all domains, and can therefore be general enough to be applied in many contexts. In the discussion of the objects situated at Novell's boundaries that will follow in chapter 5, I will show that such a symbolic object is important for creating a symbolic unity among the collaborators of openSUSE. By drawing upon Durkheim's notion of symbolic totems, I will argue that this type of object may be even more potent than Star and Griesmer might have imagined – because they partially explain how Durkheim's organic solidarity is held together. We will return to this issue later.

First, we need to find a theoretical framework that can be used to describe our two collaborative groups: Novell and the openSUSE community.

Autopoietic organization theory

The theory of boundary objects has proven to be fruitful for studying interdisciplinary or boundary-crossing collaboration. To identify and describe the collaborating groups, this theory often draws upon an understanding of social worlds (Strauss, 1978) or communities of practice (Wenger, 1998) as the social units of analysis. In the case of firm-sponsored communities, these theories have difficulties with differentiating between the unique features of a commercial sponsor organization to that of an open source community. Novell, as a traditional organization, represents a closed system of decisions compared to the open and interactive openSUSE community. Neither the interactionist theory of social worlds nor the idea of communities-of-practice is able to capture this distinction. I have previously argued that the concept of social worlds, on the one hand, might be misleading in efforts to identify various groups in collaboration, as they ultimately in fact share the same social world unified by an organic solidarity. The idea of communities-of-practice, on the other hand, is a fine-grained theory that would be more useful in identifying multiple communities within each of the interacting groups, at a more detailed level of analysis (see for example Brown and Duguid's discussion of organizations as communities-of-communities (1991) for more on this). A search for an alternative framework is therefore required.

We are looking for theories that may include more structural aspects at the right level of analysis (organizational level). Within organizational theory, one of the largest contributors on this area is Henry Mintzberg, whom has developed a descriptive framework on organizational structure and strategy based on the compilation of an impressive amount of empirical cases (Mintzberg, 1983, 1989). He develops a terminology for describing the essence of organizational structure, by identifying various groups present within all organizations, the coordinating mechanisms between them and various design parameters (such as job specialization and unit sizing). By analyzing many cases of organizations, Mintzberg finds that there are certain configurations of organizational structure that reoccur. He defines these as classical types of organizations, such as the *machine bureaucracies*, *professional organizations*, *simple structures* and *diversified organizations*, to name some of them (for more cases and details of their characteristics see (Mintzberg, 1989, p. 110)). Surely this typology and the characteristics related to them could be applicable in describing Novell, at the appropriate level of analysis. However, is such a theory also able of capturing an organizational form as extraordinary as an open source community, as present in the openSUSE project? Mintzberg has himself not provided a specific typology for this type of distributed organization, which relies heavily on information technology for operations and for

mediating communication. However, the research of Lars Groth has taken Mintzberg's work further (Groth, 1997, 1999). In an effort of clearing up a mess of organizational typologies and buzzwords in the management literature on new organizational forms, Groth has examined which new organizational configurations that in fact *have* emerged due to the evolution of information technology. Drawing upon theories of physiology and cognition while investigating the new technological capabilities, he finds that information technology in most cases only strengthens the existing organizational configurations identified by Mintzberg. There are however two exceptions and examples of new forms of configurations. One of these is what he labels "The Organized Cloud", which has characteristics that are promising in describing an open source community (see Groth 2001 for a closer description).

We need a theory that can simultaneously focus on process while maintaining a structural perspective. In sum, Mintzberg's framework with Groth's extension may seem fitting for characterizing Novell and the openSUSE project. Although Mintzberg's theoretical framework is rich with concepts related to strategy, process and structure, and could be highly relevant for our case, I nevertheless believe that we should look for an alternative theory. Instead, I will draw upon the German sociologist Niklas Luhmann and his understanding of *autopoietic social systems*, for several reasons. First, it is interesting to use a theory that has only recently received attention in organizational science, and which potential seems under-explored. Secondly, Mintzberg's theory might be more adept at analyzing formal organizations and less suited for describing ambiguous and semi-organized groups such as the openSUSE community, which has a rather unclear identity. Luhmann has a minimalist understanding of social systems. In his view, they are only based on communicative events. As we will see, this perspective is very helpful in order to identify the boundaries of the group of external developers that collaborate with Novell, which is otherwise quite difficult to frame. The remainder of this chapter will therefore present and discuss Luhmann's theory, and hopefully show how it may be useful for us.

Introducing Niklas Luhmann

The social sciences - and sociology in particular - has been the vantage point of numerous theories on the life of organizations, perhaps starting with Weber's famous model of the bureaucracy. By drawing upon general theories of social action and society, many theorists have applied such knowledge to this organized realm of social life. The German sociologist Niklas Luhmann has done something similar. Although not as famous as Weber, Luhmann has produced a comprehensive social theory with an overwhelming amount of publications that may match those of his German colleague in numbers. However, a majority of his work has never been translated to English, and so the significance and use of his theories within the English speaking scientific community does not match the volume of his academic production. Luhmann's theory is also considered quite controversial in the realm of sociology. Having been a

student of Talcott Parsons, his starting point is that of functionalist sociology. Luhmann's ideas draw upon a framework of *systems theory* for understanding the social world, and he goes as far as claiming that these systems have a real world existence – a notion that is strongly rejected by interpretative sociology. John Mingers (2003, p. 107) thus labels Luhmann a radical functionalist. However, Luhmann's theory also contains elements that are completely ignored by and perhaps even contradict functionalism. For example, rather than viewing social systems as stabilizing and harmonious, Luhmann's theory addresses how change may come about in such systems. Furthermore, Luhmann's focus is not on structure, as we might find in Parsons' work, but on systems. Perhaps the most interesting aspect of Luhmann's work is his focus on social systems as *autopoietic* – an interesting concept borrowed from biology. This understanding sees systems as recursive and self-reproductive, and emphasizes the continuous evolution of a system based on previous events and processes. Luhmann's theory is therefore able to balance process *and* structure (Bakken & Hernes, 2003), along with theorists such as Latour (1988) and Giddens (1984) - although they all do it in a different manner. The duality between process and structure makes Luhmann's theory promising for describing the ongoing evolution at Novell.

Luhmann's theory is first and foremost a general theory of social systems, which he himself applied to a number of areas such as economy, politics, law, culture, etc. Towards the end of his career Luhmann applied his theory more specifically on organizations (Luhmann, 1988, 2000). An interest in this work has recently and increasingly been picked up by organizational theorists. In 2003, Tore Bakken and Tor Hernes published a collection of contributions to organization theory based on Luhmann's understanding of autopoietic social systems (Bakken & Hernes, 2003), and David Seidl (2005a; Seidl & Becker, 2005b) has used his theory to particularly address the topic of organizational identity. Michèle Morner (2003) has shown how Luhmann's theory may be particularly suited to describe the emergence and stabilization of open source software projects, which again makes this theory even more suitable in the context of this study. In the following I will start off by reviewing Luhmann's social theory, before turning to how it may be applied to organizations in particular. I will also discuss some issues that I find problematic with this theory.

Features of autopoietic social systems

The basis of Luhmann's theory starts with an understanding of the social world consisting of *social systems*. A system is typically defined by a boundary between itself and the environment. Whilst the environment may be infinitely complex, systems create order that reduces this complexity within them. These systems, in Luhmann's view, have a *real world existence* (Bakken & Hernes, 2003, p. 10). Not only do they serve as an analytical tool for studying society, they are actually tangible entities. If they were removed, the social world would disintegrate. The social systems are as real as our physiological and psychological systems, which Luhmann holds in

parallel. What distinguishes social systems from these other types of systems is that social systems produce and process *meaning* based on communication (Mingers, 2003, p. 104).

For Luhmann, social systems are made up of a network of *communications* (Mingers, 2003) . Communications are the micro-fibers of any social system, and constitute the core of Luhmann's theory. He uses the term communication in a quite specific sense. An instance of communication consists of three basic elements: (1) information, (2) utterance and (3) understanding. *Information* refers to the content of the message, while *utterance* is the form in which it is sent and the intention of the sender. The receiver will then create an *understanding* of the message, an interpretation that may well differ from the intention of the sender. After these three stages of the communication are completed, the receiver may then accept or reject the communication's *meaning*, and this is the link to action and what enables further communications. The three elements form a unity. Unless all three elements are present, it is not communication in Luhmann's sense. For example, an email that is sent but is devoured by spam filters and never read by anyone is not a communication - no matter how important the content of the email or intention of the sender. Other examples of communications may be the purchase of a car within the economic system, or the casting and registering of a vote within the societal system. The basic elements of a social system are therefore not thoughts, behavior or actions, but the network of *communications*.

A central feature in Luhmann's understanding of systems is that they are *autopoietic*, meaning that they are self-referential and self-reproductive. Systems constantly create and recreate their own elements and boundaries towards the environment. A system cuts itself off from its chaotic and uncontrollable environment in creation of these boundaries. Bakken and Hernes claim that "[t]he idea is compatible with that of recursivity: an autonomous or operationally closed system reproduces its own operations by the network of its own operations." (2003, p. 11).

Communications within a system are therefore always the result of *previous communications within the same system*. For example, you know that you can buy a car at the local car dealer because it has been done before. The director at the car dealer may change the sales-price for a car since he has been given this authority by previous communication from the board. This means that every communication that is made also *enables new communications*. "[T]he autopoietic process is thus the continual generation of communications by communications" (Mingers, 2003, p. 107). All communications are linked by the meaning they process and create. If these communications stop, for whatever reason, the system dies.

The term autopoiesis (pronounced "auto-poy-E-sis"; literally: self-creation) originates from cognitive biology, by thinkers Humberto Maturana and Francisco Varela. They developed an alternative to Darwinian ecology theory, where their key argument is that biological systems are

not subject to environmental selection. Rather, systems are closed off from the environment, enabling them to interact with themselves (Bakken & Hernes, 2003, p. 13). Systems draw upon resources from the environment, but these resources do not become part of the systems operation. That the system is closed off from its environment does not mean that it has no interaction with the environment. Luhmann's notion of a closed system has nothing to do with the technical understanding of an open or closed system. "Social systems are communicative systems, which relate themselves to their environments." (Jönhill, 2003, p. 23). Social systems are autopoietically closed in that they use and rely on resources from their environment; yet those resources do not become part of the systems' operation. For example, people are not part of a social system – they belong in the system's environment. It may sound like a huge paradox to exclude people from a social system, but a system in Luhmann's reductionist view only includes the operations that constitute its structure. The operations of a social system are communications. Of course, people may be performing the communications, but it is only the communication itself that are part of the social system. The person belongs to other systems; the body being part of the physiological system as the cognitive aspects are part of the psychological system. In this way Luhmann consistently avoids confusing domains in his systems theory (Mingers, 2003, p. 112).

A key feature in Luhmann's theory is his emphasis on *events*, which is important for explaining the relationship between process and structure in his systems theory. Every communication is an event. The autopoietic structure is produced and reproduced in a *process* which consists of a *chain of such events*. It is therefore important to acknowledge that every structure has an important historical trajectory that explains its current condition and future possibilities. When we are to describe organizations and systems in Luhmann's sense, we therefore use *communicative events* as the basis of our analysis instead of focusing on people. As we will see, this counter-intuitive understanding may actually be very useful for us, especially in the case of describing the active contributors in the openSUSE community and the boundaries that surround them. It is important to note that although people are not the focus of Luhmann's systems analysis, we may however need to study people methodologically in order to trace the communicative events that are created by them.

Interactive, societal and organizational systems

Luhmann distinguishes between three different types of social systems that have evolved in modern society. *Interactive systems* are the most basic systems of communication which may involve as little as two parties. They are the most fundamental of systems, and usually involve people being physically present with each other so that they may communicate directly with each other and even read meaning from each other's facial gestures. The systems boundaries are defined by the meaning and nature of the communication. "The boundary of the system is constituted in terms of those who are included within the interaction, those who can be spoken *with*, as

opposed to those who are excluded, those who can only be spoken *about*.” (ibid, p.104 – my emphasis). Interactive systems emerge when several individuals engage in related communication and perceive the fact that they are engaged in communication (Morner, 2003). When presence ends, the system ends. However, as Morner argues, physical presence may not always be required, as technology and the internet in particular has enabled communication among geographically distributed participants (2003). I would also argue that an interactive system can have a duration of a long period of time. For example, I would believe that an academic discourse (such as the debate of positivism in social science) may be regarded as an interactive system.

Societal systems are on the opposite side of the scale, and involve a more complex web of communications. Luhmann generally refers to only one single societal system, the all-encompassing world system. Interaction within the societal system relies on the existing societal structure of communications and expectations. At the same time, communications contribute to society’s reconstruction. In this way Luhmann’s theory has similar features to Giddens’ theory of structuration (Giddens, 1984). The societal world system may further be divided into subsystems. For example, each nation may have its own societal system that differs from the other nations. Luhmann also identified several societal systems in different domains, such as law, economy, science, mass media and politics. In his review of Luhmann, Jan Inge Jönhill refers to these subsystems as *functional systems* in society (2003).

Organizational systems are a specialized form of social system. Organizations are more fixed networks of communications where the boundaries are somewhat clearer. “The closure of such systems is shown by the fact that, in general, communications within the organization are about things happening within the organization, or are *about* the environment but not generally *with* the environment” (Mingers, 2003, p. 109) - emphasis added). Luhmann does not differentiate between different types of organizations, i.e. between firms and public institutions, because they all share the same general characteristics. The most characteristic feature of organizations – as opposed to other social systems – is that their communicative events consist entirely of *decisions*. Again, Luhmann has a specific understanding of this term. Decisions are a specific form of communications, where selection is made between various options. They can be understood as *events* that produce a *transformation of contingency*:

“Before the decision, several possible decisions exist, thus the space of open possibilities is limited. After the decision, the same contingency exists in a fixed form: the decision could have been made differently – it is now contingent upon itself...” (Luhmann, 1988, p. 37).

Decisions absorb or reduce uncertainty in organizations, but they also create new uncertainties. For example, if a decision is made to create a product, new questions arise: How will we market it? Who will buy it? Where do we produce it? In this way the network of decisions create the need

for new decisions, and all decisions are also based on previous ones. For example, which decisions John can make in the company is influenced by a previous decision of appointing him as regional sales-manager. This is the autopoiesis of organizational systems.

Drawing upon the words of Herbert Simon, Luhmann identifies three different forms of “premises for decision-making” within an organizational system. These are important in understanding the underlying structure of decisions within an organization, and offer a framework that relates to a more traditional understanding of organizations. The three major premises within an organization are *programmes*, *channels* and *personnel*. *Programmes* are the “procedural recipes” in the organization that provide guidelines as to which decision should be made in a given situation (e.g. if situation equals A, do X). “The organization itself will tend to regard the programmed correctness as rationality” (ibid, p.45). Decisions that contradict these programmes may also be understood as rational by the organization, as they become categorized as exceptions from the rule. The second premise is the establishment of *channels of communication* (and the exclusion of others) in the organization, which “permit the circulation of information with a binding impact on the system” (ibid, p.46). They facilitate and provide a structure for communications which Luhmann labels the “communication web”. This web may be formed as a result of the division of labor, but does not necessarily follow a hierarchy from top to bottom. The third premise involves the *personnel* in the organization who “participate in the decision-making process with body, mind, reputation and personal contacts, and in this way partly extend, partly limit what may be decided” (ibid, p.46). The people in the organization - or rather the roles they occupy – determine which decisions they are able to make.

Put together, these three premises – decision programmes, communication channels and personnel - link together in a network of “*positions*” (Mingers, 2003, p. 110), where “[e]very position may be regarded as a combination of programmatic, web-like and personal decision-making premises” (Luhmann, 1988, p. 47). Each position thus follows certain decision programmes, has certain channels of communications (who they talk to and how) and has personnel that occupy them. Luhmann admits that “[t]he number of positions defines in an abstract way the size of the organization” (Luhmann, 1988). This brings us to the last characteristic of organizations, which is Luhmann’s recognition of membership:

“Establishing organizations presupposes a rule of recognition permitting the system to establish which acts are valid as decisions in a system, and under what aspects they may be referred to as such. This recognition rule is above all a membership rule. It establishes who may be referred to as a member of the system and in what roles this membership may be exercised. It is always concerned with a role specific definition, not about the inclusion of the collective posture of an actual human being in the system. Through a selection of individuals and definition of roles, the system may differentiate itself in ways that cause (...) recognition for itself and others” (Luhmann, 1988, p. 38).

An organizational system thus recognizes - through its decision-network – who are members of the organization. As this excerpt illustrates, it is not the people *as such* that are members of the organization, but the roles the individuals occupy. It is therefore possible to identify which people have roles as members of an organization, and which do not belong within the system.

The three forms of social systems – organizational, interactive and societal systems – may overlap each other. For example, there are many organizations that span across several functional systems, and that may operate in the realm of law, economy and politics at the same time (Jönhill, 2003). In fact, such organizations are absolutely necessary in order to reconcile these independent functional systems.

Criticism to the theory

At a general level, Luhmann has received a fair amount of criticism for his theory, most notably from fellow countryman Jürgen Habermas about the potential of social systems theory. Luhmann follows the tradition of his mentor Talcott Parsons in creating a “grand theory” of society, a theoretical approach that has been severely criticized for being too little in touch with the empirical world. Luhmann’s theory has also been criticized for being highly abstract, and his publications for being difficult to read. A major critique is found in Piyush Mathur’s review of Luhmann’s theory of Ecological Communication (see “Neither Cited nor Foundational: Niklas Luhmann’s Ecological Communication” (Mathur, 2005). A more general critique of Luhmann’s theory of social systems is given by Alex Viskovatoff (1999).

At a more specific level John Mingers (2003) targets some weaknesses in Luhmanns theory. First of all, Luhmann does a poor job of relating his social systems theory back the social being – the human individual. As we have seen, it is previous communications - and not people - that create new communications in Luhmann’s sense. Humans are therefore largely ignored, although it is clear that there would never be any communication without them. Focusing on communications and not people may nevertheless have some advantages, in this author’s opinion. For example, it makes it possible to differentiate between whom, within a group of passive members, actually part-take in a social system, by separating those participating in communications from those who do not. I will return to this point in the discussion of the openSUSE community later.

Mingers also addresses a second problem: Luhmann’s more or less functional approach completely ignores the more interpretive, action-oriented aspects of organizations. Luhmann sometimes refers to the autopoietic psychic systems (i.e. people’s consciousness), but fails to couple this with the social communicative system. The link between the two is apparently Luhmann’s notion of meaning, but this is also a somewhat unclear concept in Luhmann’s theory.

I would also add some additional comments concerning organizational systems, as there are several issues that are unaccounted for in Luhmann's theory. In general, Luhmann has a very reductionist view of organizations. His definition of organizations as a network of decisions largely ignores and fails to explain organizational processes and the influence of organizational culture, which modern organizational theorists give a much larger role. Luhmann's theory also fails to explain how decisions are made, or rather, which decisions should be made in situations where there is no program to prescribe a specific behavior. Thoughts people might have and actions they make are usually not part of this decision-network (they belong to the environment), although they might *trigger* decisions. Keeping them external to the system thus black-boxes much of the decision-making process. This critique coincides with other remarks on Luhmann's theory, addressing its failure to deal with *normative* issues. One of Vistovotoff's main concerns is that since Luhmann's theory is purely descriptive, and has no explanatory power (Viskovatoff, 1999). Luhmann would most likely reply that this is no accident. Luhmann intentionally sought to produce a descriptive theory of society following the footsteps of Max Weber's non-normative science, later re-defined and defended by Karl Popper. Luhmann's theory therefore does not make any claims about whether organizations are driven by goals or values, or what rationality they may follow. He neither denies that these exist, or that organizations have identities and culture (Luhmann, 1988). Admittedly, keeping things simple also has some clear advantages. Reducing the organizational complexity by focusing on its core – the network of decisions – can make them easier to analyze. There are a lot of things going on in organizations that may not affect the evolution of the organization itself, and actions, communications and people that are only loosely affiliated with the organization may therefore be excluded from the analysis.

Another issue that may be problematic in Luhmann's understanding of organizations is his notion of *closed* autopoietic systems. Since the emergence of contingency theory in organization science, closed and autonomous systems have been considered a theoretical dead-end within this line of thought. Contingency theorists such as Joan Woodward (1958) claim that organizations are influenced by various aspects of the environment: the contingency factors. Luhmann is clear on stating that organizations are only influenced by their own decision-network, and he is thus a fair target for criticism from such theorists. I however find it necessary to clarify that Luhmann's understanding of autopoietic closure does not mean that a system is unaffected by what happens in the environment, or that the system fails to affect the environment they operate in:

“...autopoietic systems dispose of a recursively enclosed mode of operations. But they are neither non-environmental systems, nor can they operate without having an impact on and through the environment. In the context of autopoietic reproduction the environment functions as irritation, disturbance, noise and can only become meaningful when having an impact on the decision-making context of the system” (Luhmann, 1988).

When a condition in the environment makes it necessary for an organization to respond, the organization makes this condition a part of its internal decision-network. An example, provided by Luhmann, is that an organization may expand its *variety* of decisions (e.g. moving into new markets) as a result of turbulence and structural changes in its environment (Luhmann, 1988, p. 42).

Open source software projects as social systems

Despite its shortcomings, Luhmann's theory of social systems and organizations has some interesting features that deserve attention. Michèle Morner (2003) has looked specifically at how Luhmann's framework can be used to understand open source projects. She claims these provide a figurative example of organizational emergence, as groups of collaborating individuals form around specific software projects. Some of these groups evolve into stable organizations while others wither away and disappear. She asks two questions: What kinds of organizational form do these emerging projects (with no formal members and continuously changing participants) represent? And what are the circumstances that influence the stabilization or disappearance of emergent open-source software projects?

First, Morner suggests that open source projects can be understood as *interactive systems* (described above). Although the participants are not physically present with each other, they are present in the form of perceiving each other via the Internet and have the possibility to follow each and every communication because of its documentation in mailing-lists or newsgroups. An interactive system also needs an attentive core, a subject that sets the boundaries for the interaction of the participants. Morner suggests that the modular structure of the open source software projects provide them with a subject-structure that naturally binds the communication between the participants.

Morner argues that some open source software-projects may also be more typical of *organizations* than of interactive systems, based on the fact that decisions occur and decision programmes exist in many large software projects. Also, many open-source software-projects communicate outwardly via a project leader. Morner claims that this is the only way the environment can recognize open-source projects as systems, and that this would not be possible for interactive systems. This kind of open source communities may also fit the characteristics of what O'Mahony describes as community-managed projects (2007b). In the case of openSUSE we will be looking at a community that is *not* entirely self-managed, and will find that the first category therefore may be better suited.

Based on Luhmann's systems theory, Morner argues that certain premises contribute to the stabilization of emergent systems. These are factors that discern the projects that vanish from

those that become more or less permanent. The first group of factors concerns the *connectivity of communications* – the inherent potential of communications to be perceived and succeeded by other communications. There are two factors that influence this connectivity: the *accessibility* of the communications and their thematic *modularization*. In open-source projects, both of these criteria are fulfilled as most communication is open for everyone on written channels and are organized by topics. The second group of factors concerns the *systemic memory* of the system, and consists of documentation & archiving and decision-making programmes. Morner argues that all communications within open-source software projects are reproducible at all times, as they largely exist in mailing list archives. Communication regarding the development of the source code is also for the most part archived. She also argues that a decision-making framework is provided through various formats (such as bug-formats and hackerguides) and voting systems for making decisions on changes in source code. We may therefore conclude that Lumann's theory can be suitable to describe this kind of organization that openSUSE may be an example of.

~~~~~

We have now reviewed the theories we will use for this thesis, and are ready to move on and study the empirical case. Be aware that we will need to let Luhmann's social systems and Star's boundary object rest while we go into the world of Novell and openSUSE. We will bring them back and discuss Novell's collaboration with the openSUSE community in light of these theories in chapter 5. However, before we can even start the empirical journey, I will first present and discuss the methodology I have used to guide my data collection.



## Chapter 3

# Methodology

---

In a research project, many different strategies may be used to align the relationship between data, theory and research objectives. For example, one may base research questions entirely on a theoretical framework and seek to 'apply' this theory on a given empirical reality, in a deductive manner. On the opposite side of the scale, one might move into the field with nothing but blank sheets of paper and generate theory inductively based on the empirical constructions one finds – sort of creating a theory based on the words your informants (grounded theory is an example of such an approach (Glaser & Strauss, 1968)). This chapter will present the approach and techniques I used to study Novell and the openSUSE community. Towards the end of the chapter I will turn to discuss how much we can trust the findings and conclusions in this study, based on our knowledge on the use of these methods.

### ***Research strategy***

My own research strategy is tightly coupled to the empirical nature of my research questions. I have therefore largely been guided by an inductive, qualitative approach using ethnographic methods. To strengthen and supplement this approach I have used some quantitative techniques that enrich my findings from interviews and observations. Although this strategy was to a large degree outlined before the actual fieldwork began, it would have been impossible for me to describe exactly what I would end doing before it was done. In fact, this chapter has largely been written towards the end of the work on this thesis for this very reason. This is not uncommon for qualitative work: “The beauty of qualitative research is that its rich data can offer opportunity to

change focus as the ongoing analysis suggests.” (Silverman, 2005, p. 80). This is in particular an appreciated privilege when investigating a phenomenon that is largely unknown.

At the early phase of this research I was stressed by searching for the perfect theory to guide my data collection. Being frustrated for not finding such a thing beforehand, I was comforted by some personal advice offered by Steve Barley, an experienced ethnographer at Stanford University: In much qualitative work the right theory to describe your data is near impossible to find until you have the data in front of you. This is due to the fact that you seldom know what you are looking for before you have found it (although you of course will have an idea of which direction you will be looking). Finding a match between theory and data has therefore been an ongoing process of testing, investigating, modifying and discarding theories between my advisor and I – a form of *throw-away prototyping*, to use an analogy from computer science. In other words, I did not know that what I was searching for was Luhmann’s autopoietic systems while I was gathering data in Novell. Rather, I was searching for rich descriptions from the field and *a posteriori* investigating which theory might make most sense of this reality. However, to say that my research strategy for this study has been purely inductive without guidance of any theory, would also be wrong. Although I have not been driven by any hypothesis as such, I have unavoidably been driven by my own assumptions. Theories from the field of open source software development along with the theory of boundary objects and communities of practice were also lingering in the back of my head as I conducting my interviews and observations. I have tried to use these assumptions to guide me throughout the research and point me in the right direction. I find that Ragnvald Kalleberg gives a precise account of the process a researcher wanders through. He describes it as a constant inter-relational negotiation between questions, theory, data and answers. All elements mutually impact each other throughout the entire project, as illustrated in figure 5 (Holter & Kalleberg, 1996).

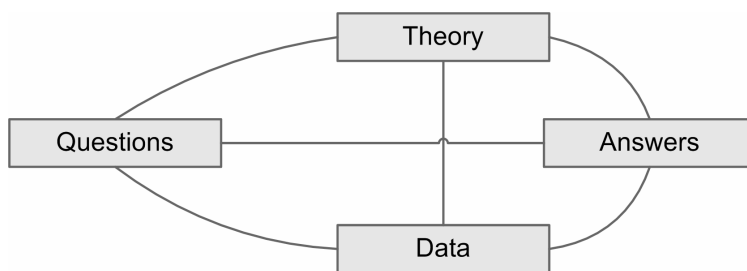


Figure 5. Elements of a research project. From Holter & Kalleberg (1996, p.33)

The interdependent nature between these four elements provides an accurate description of the process I have undergone in this project. As mentioned, the data I gathered influenced the theories I chose to describe the empirical reality. At the same time, my theoretical understanding of the dilemmas of firm-sponsored open source communities also guided my focus in the

interviews and therefore my data collection. More than anything, the answers I found underway gave rise to new questions which again impacted my research design and methodology. For example, I had not originally planned to include a quantitative approach in my research design, but during the course of my interviews new questions started to arise concerning the openSUSE community: Who are they? What are their normative sentiments towards Novell? Are they a group or just individuals in a network? How do the Novell employees impact the activity in the external community? I found that I could not get fully satisfactory answers to questions such as these through my original design, and I therefore chose to include a survey and study some mailing list statistics. In the following, I will detail all the methods I ended up using for this project.

## **Research design**

### **Case study**

I have chosen to select a *single case* for my study, which is Novell and their openSUSE project. Using case studies is a common strategy in qualitative research, where the basic idea is to study one case in detail using whatever method that seems appropriate. The general objective is to develop as full an understanding of that case as possible. This study provides quite an extensive coverage of Novell's open source strategy and their collaboration with the openSUSE community. My choice has thus been to say "a lot about a little" rather than the other way around, which is the sound advice Silverman offers for all qualitative research (2005). I chose to use *one* case since my research questions demand a detailed investigation of organizational processes. I would not have had the resources to investigate multiple cases at the level of analysis my questions required. Also, by focusing on one case it was easier to go as deep as possible and 'turn every stone' in the search for relevant data that could provide answers to the research questions.

At an early stage of this study I did however consider using a broader empirical approach, by doing a comparative study among a number of cases that fit the characteristics of my theoretical case (firms with their sponsored open source communities). I chose not to do this for the reasons stated above, and because there already exists some comparative research on cases from this population. O'Mahony's research (2007a) shows that firms with sponsored communities struggle with balancing control and openness in product development (as presented in the previous chapter), but does not detail how this balance is achieved. Theoretically and empirically it is therefore more interesting to go deeper into one case to search for detailed answers. In the following I will present the methods that are used to gain this knowledge.

## Interviews

The backbone of my empirical data consists of interviews with employees in Novell and members of the voluntary openSUSE community. In total I interviewed 26 people, not including informal conversations with informants at various events. All interviews were open-ended, and lasted on average about 45-60 minutes. For each interview, many of the questions I prepared were altered or modified beforehand based on what I had learned in previous interviews. I started off with an interview guide, but it was continuously changed for each interview. All interviews were done face to face, except for the interviews with community members across the globe (these were done by phone). Most interviews were transcribed and analyzed in-depth after all field work was done. A complete list of all informants is provided in table 3 below.

| Nr | Position                        | Date                         |
|----|---------------------------------|------------------------------|
| 1  | Novell consulting manager       | 18.05.2007                   |
| 2  | CEO - Cleversafe                | 24.05.2007                   |
| 3  | CEO – JasperSoft                | 24.05.2007                   |
| 4  | CEO – MySQL                     | 25.05.2007                   |
| 5  | Novell marketing manager        | 25.05.2007                   |
| 6  | Former Novell strategist        | 25.05.2007                   |
| 7  | Representative – Hyperic        | 27.05.2007                   |
| 8  | Novell openSUSE product manager | 10.10.2007                   |
| 9  | Novell openSUSE manager         | 11.10.2007                   |
| 10 | Novell openSUSE engineer        | 11.10.2007                   |
| 11 | Novell openSUSE manager         | 11.10.2007                   |
| 12 | Novell openSUSE manager         | 11.10.2007                   |
| 13 | Novell openSUSE engineer        | 15.10.2007                   |
| 14 | Novell openSUSE director        | 16-20.10.2007 (3 interviews) |
| 15 | Novell openSUSE engineer        | 17.10.2007                   |
| 16 | Novell openSUSE engineer        | 24.10.2007                   |
| 17 | External community member       | 25.10.2007                   |
| 18 | External community member       | 25.10.2007                   |
| 19 | Novell openSUSE engineer        | 25.10.2007                   |
| 20 | Novell openSUSE engineer        | 25.10.2007                   |
| 21 | External community member       | 26.10.2007                   |
| 22 | Novell openSUSE engineer        | 26.10.2007                   |
| 23 | Novell openSUSE engineer        | 26.10.2007                   |
| 24 | External community member       | 27.10.2007                   |
| 25 | External community member       | 27.10.2007                   |
| 26 | Novell openSUSE manager         | 19.11.2007                   |

*Table 3. List of interviews.*

The list is sorted chronologically by the date of the interview. In the following chapters the informants are cited and referred to by their number on this list (e.g. interview # 13). I have included the date of the interview in the short-hand reference only in cases where the interview is done *before or after* the main data collection in October 2007. Although several informants offered me to cite them by name, I have held all references to names anonymous. Although many of my informants will say they do not need such protection, it is both practical and avoids the risk of any ethical concerns.

I have not used extracts or cited all of the informants on this list, and the observant reader will notice that a few key informants are cited more often than others. However, I have gained knowledge from my informants. Since they all have influence the study, they are therefore included in the list. For example, although I chose not to pursue a comparative study, the interviews with the other open source companies were important for understanding some of the dynamics of open development better. They also provided a better understanding of Novell in contrast to these other companies.

### **Ethnography**

I stayed with the Novell's Linux R&D for about a month in October 2007 while conducting most of the interviews. My ethnographic approach was *participatory*; I was not there to merely make visual observations, but to engage in conversations and activities with managers and developers. This gave me the opportunity to get more familiar with the company and people from the inside, and provided an abundance of information and practical help. By being present at the office every day, I became familiar with managers and engineers and learned to distinguish what information was more important. I took field notes for every day I was present. In the evenings I would often participate in social events. The observations I made from informal settings such as these and in coffee breaks at the office were immensely important for making sense of the discoveries I made and information I gathered. By engaging in interesting conversations at restaurants and cafés I also discovered new topics and questions to pursue in my study. All in all, these observations were very important for the study, and I would again like to thank Novell's Linux R&D for their hospitality and help.

Other ethnographic experiences have also been important for this study. For example, participating at the Open Source Business Conference in San Francisco (May 2007) was quite a contrasting experience in comparison to a small community conference in Germany (Practical Linux 2007, Gießen). I have also been following and observing discussions on some of the openSUSE mailing lists for more than 5 months.

## Simple statistics

During my stay at Novell's R&D team in Germany, I learned that the mailing lists and IRC channels (described later) were important indicators of activity in the openSUSE community. I therefore decided to gather some statistics on this activity by examining email archives that were publicly available on [www.opensuse.org](http://www.opensuse.org). I wrote a software program<sup>7</sup> that was able to read the name of the sender of every message and compile some simple statistics (such as calculating frequencies and percentages). These data are only used at an aggregate level, and the information that is used can not be identified with any single person.

## Survey

While conducting my qualitative research and compiling the quantitative statistics, I became increasingly interested in finding out more about the normative sentiments in the external openSUSE community. I knew that I would be able to speak directly with informants from the community, but I also wanted to find out how heterogeneous or homogeneous the community was in terms of their individual background, motivation, opinions and sentiments on certain issues. Since I could not rely on single informants to gather this kind of knowledge, I conducted an open questionnaire-survey among active community members. This proved to be a source of information that could extend and confirm my understanding of the discoveries I made in the qualitative research. In the following I will make some notes about the validity and reliability of the survey, and about some of the statistical measures that have been used.

The survey targeted *active contributors* in the *openSUSE community*, and 281 people participated in the survey. Due to statistical methods, this number of participants in a survey would normally be enough to render the findings in the survey applicable to the entire community. However, this requires that the selection of participants for the survey is also done according to statistical methods (i.e. randomly selected among the entire community (Ringdal, 2001, p. 143)).

Unfortunately, the selection of participants for this survey has not been done according to such methods, for two reasons: I was not given the liberty to use individual contact information by Novell to target randomly selected participants directly, due to reasons of confidentiality and because these contact details had not been authorized for such use by the individuals beforehand. Second, in order to create a random statistical sample group, a clear definition of the entire population is needed. Since the boundaries of the active community are unclear (see the discussion of this topic in the next chapter), it would have been hard to determine exactly which individuals to include in the selection of participants. Instead, this survey has been conducted with an open invitation and is based on voluntary participation. The invitation was sent directly

---

<sup>7</sup> The program is written in Java, and is available by contacting the author. Many thanks to Martin Lasarch at Novell for providing valuable help with the Grep command line tool.

to the following mailing lists<sup>8</sup>: project, factory, buildservice, translation, wiki and the main list (opensuse@opensuse.org), which are the most important channels for reaching the active contributors, according to my informants. In the invitation, members that contributed actively to the project were encouraged to participate. In this way it was made the choice of the community member to define his/her own status. A link to the survey was also posted on news.opensuse.org. As it is very likely that active contributors in the community follow one or more of these communication channels, it is also likely that the invitation reached a large majority of the target group.

The lack of statistical randomization of the respondents in the survey means that the results of the survey cannot be statistically generalized to the entire population (all active contributors in the openSUSE community). I will therefore advise the reader to be cautious concerning the reliability of the findings in the survey. The survey does however provide some value: First, the results are definitely valid for the group of people who answered the questionnaire (meaning the 281 participants whom voluntarily participated). According to my trusted informants (openSUSE engineers and managers), the target group of active contributors in the community is not estimated to be much larger than this (although this may depend on the definition of target group). Furthermore, the answers to the survey confirm that most respondents fit the understanding of an active contributor (they provided answers to how many areas and hours of contributions they put in). We can therefore say that it is *likely* that the findings in the survey are somewhat representative of the group of contributors in the community, but the results may not be statistically generalized and can thus not be given too much substantial weight.

### **Statistical measures**

The results from the survey are for the most part shown as pure distributions of frequencies (i.e. a count of how many answers there are to the various categories of a question). In the analysis that follows, I will also present some correlations in the data, where this has been found significant through statistical analysis. Correlation between two variables means that they are related somehow in the data (for example, the more X increases, the higher value of Y). I will mainly use the statistical measure Gamma to determine the strength of the correlation, since this measure is suitable for variables on the ordinal level. It ranges between -1 and 1, where 0 means no correlation at all, and -1 or 1 is a perfect correlation (linear relationship). Negative and positive numbers indicate the direction of the correlation. All correlations I present from the survey are statistically significant at the 0,05 level, meaning that the risk of reaching the results by coincidence is less than 5% (most findings presented here are in fact significant at the 0,01 level).

---

<sup>8</sup> A description and overview of these mailing lists is provided in the next chapter.

There are several common pitfalls in using correlation. Correlation is symmetrical, not providing evidence of which way causation flows (one can not say if is X or Y that “determines” the other). If other variables also cause the dependent variable, then any covariance they share with the given independent variable in a correlation may be falsely attributed to that independent. For example, if we believe that X is the cause of Y, then we can not tell if there is another variable Z that is in fact equally or more responsible in explaining Y. I will however express my interpretations clearly where correlation is used, so that the reader is able to critically assess my reasoning.

## **Reflections**

There are two important questions to discuss concerning the strength and weaknesses of the research approach detailed above: Based on the chosen approach, can we trust the results we find with these methods? Second, how can we know that the conclusions that are drawn from these data are true? The goal is to ensure that the methods used are *reliable*, and that the conclusions are *valid*. I will discuss these two aspects in turn below, but first I will make a note on the benefits of triangulation and discuss the problem of generalizing the findings in qualitative research.

## **Triangulation**

In my approach I have used multiple methodological techniques together. This is often referred to as triangulation, and can offer some potential benefits. In my case I have seen that combining methods has given some synergies. The most important benefit is strengthening the validity of the findings by confirming them with different techniques. For example, I heard from openSUSE managers that external community members did not identify themselves with Novell – only the openSUSE community. The survey clearly confirms this, as only 10% of the respondents to a large degree think of Novell and openSUSE as one and the same. There are also examples where other data sources have falsified findings. For example, a couple of my informants in the company told me that a specific controversial deal made between Novell and Microsoft in 2006 no longer was of any concern for the outer community. And although they might be right in the fact that the ones who were very upset by now had left the community, about half of the respondents in the survey did indeed still have concerns related to this deal.

Another benefit of triangulation is that the various techniques can supplement each other. For example, my interviews with community members were important for preparing the community survey. The interpretation of the results from the survey were also enriched by the qualitative descriptions I could draw upon from interviews and observations. Another example includes using observations for preparing interviews. For example, participating in informal settings after working hours was very important for discovering interesting topics to pursue later on, in formal interviews. Over a meal and some drinks, interested Novell engineers would ask me about my



study and talk about their opinions on various matters, and I could ask them about their interpretation of my temporary findings.

### **Generalizability**

Qualitative research – and single-case studies in particular – are often faced with the problem of explaining how findings from such research can be *generalized*. More precisely, how do we know how representative the case-study findings are of the entire population from which the case was selected? Silverman (2005) argues there are several measures that may be used for ensuring generalizability of qualitative research and case studies, including purposive sampling, theoretical sampling or combining qualitative research with quantitative methods. However, the question of how representative the findings of any research is for a larger picture will ultimately have to relate to the nature of the research questions. If the aim of the research is to develop a theoretical model, empirical generalizability may be of no interest at all. This is the case for the third of my research questions, where I seek to explore how a set of theories can be further developed with the use of my case data. The first two questions are somewhat more focused on the explaining the empirical reality that is being studied, but also here a theoretical emphasis is highly present. My research questions are, in sum, not aiming at rendering my findings generalizable to all cases *per se*. Rather, I create an analytical framework that can provide explanations and descriptions of *this case*. I do however argue that this framework may provide equal explanations and descriptions to other cases that carry similar characteristics as the one I study. For example, since my case fits the characteristics of the population for O'Mahonys research (2007a), it is likely that a similar analysis on these cases will render similar results. I will return to a brief discussion of this topic in the conclusion of the thesis, but it is sufficient to say that showing how it *can* be done in one case is enough to answer the questions in my research.

### **Reliability**

If another researcher were to do over all my interviews and observations, would he achieve the same results as I? This hypothetical question addresses the *reliability* of the data. In qualitative research this issue must be dealt with in a different manor than in quantitative approaches. In the latter, reliability is calculated by searching and accounting for all possible random measurement errors that may be present in the data (Ringdal, 2001, p. 167). Since we are not dealing with numbers but rich descriptions of reality, we are not 'measuring' anything in qualitative approaches and can therefore not account for such possible 'errors'. Rather, we are investigating and providing interpretations of a social reality. Our primary strategy for ensuring reliability is therefore through documentation: "For reliability to be calculated, it is incumbent on the scientific investigator to document his or her procedure" (Kirk & Miller, 1986, p. 72)<sup>9</sup>. I have

---

<sup>9</sup> Quoted in Silverman (2005)

therefore made an effort to ensure that all my sources of information in this thesis are traceable to a specific interview, observation note or survey result. All my interviews have been painstakingly transcribed and are possible to access if needed. The same goes for the observation notes. Where I use data from the interviews, I have in most cases included quotes to show the words I am drawing my interpretations upon. Although I might not always explore alternative explanations, it leaves this opportunity open to the reader to critically assess the basis of my conclusions. This last point leads us to the issue of data validity.

### **Validity**

Validity is another word for 'truth', and addresses the problem of how we can ensure that our conclusions and interpretations are in fact correct? "Sometimes one doubts the validity of an explanation because the researcher has clearly made no attempt to deal with contrary cases" (Silverman, 2005, p. 210). The most common critique of the validity of qualitative research is what is known as the problem of anecdotalism:

"There is a tendency towards an anecdotal approach to the use of data in relation to conclusions or explanations in qualitative research. Brief conversations, snippets from unstructured interviews... are used to provide evidence of a particular contention. There are grounds for disquiet in that the representativeness or generality of these fragments are rarely addressed" (Bryman, 1988).

Silverman presents two common assumptions on how this problem can be countered. The first is by methodological triangulation, and the second with respondent validation. In this study, both of these features are present. I have had informants read through text to give feedback on my interpretations of their reality. However, Silverman continues to say that both these strategies can be problematic, and cannot *ensure* any form of validation (see his book, chapter 14, for reasons why). Instead, he summarizes five strategies that can be better trusted to ensure validation, based on a large body of contributions to qualitative methodology. I will address each of these in turn, and assess to what degree they have been applied in this study.

The first strategy is to pursue the principle of **refutability**, meaning that we must try to refute our initial assumptions of relations between phenomena and avoid the temptation to jump on easy conclusions based on some evidence in the data. This evidence must be tested in all any manor. I must admit, I have not systematically pursued such a strategy in this study. My findings are however supported by other research findings (such as the tension between control and openness in firm-sponsored communities), and I therefore believe this offers credibility to my interpretations as well. The second strategy is to strive for **constant comparison** the data, preferably by comparing different cases in order to find consistency in interpretations and findings. I have not had this luxury since I chose to use a single case, but this principle is possible

to apply within a single case as well by searching for internal coherence between interview cases. This requires that all data exists in an analyzable form (transcribed). I would argue that I have indeed been searching for such coherence in my data, and have not been able to find any inconsistencies that threaten my conclusions in this thesis. The third principle is related to the previous strategy, and concerns **comprehensive data treatment**. This basically involves applying your generalization successfully on every “single gobbet of relevant data you have collected” (ibid, p.215). While I have not been able to find any counter-data to the humble conclusions I draw in this thesis, I am however careful with overstating the general character of these findings. I will return to this question in the last chapter. The fourth principle concerns **deviant case analysis** and making sure that the generalized findings can still be explained by unusual cases that might seem contradicting. This strategy is difficult for me to use, first of all because I only have the capacity of studying one case. Second of all, in the domain I am studying my case *is* in fact a deviant. There is already an extensive body of research on autonomous open source communities and commercial software firms (which would be Silverman’s ‘deviants’ in this study). I therefore do not experience the need to test my findings on these cases as this knowledge is already existent. The last strategy is what Silverman describes as **using appropriate tabulations**, meaning that one can use a quantitative supplement to confirm some of the qualitative interpretations. This is exactly what I have done when I have generated statistics on mailing list activity to confirm the patterns of activity that were described to me by informants beforehand. This quantitative data confirmed what I learned from the descriptions I received in these interviews, and gave me “hard data” to back up my findings. I however do not see why Silverman separates this strategy from triangulation, which this principle is only a specialization of. Therefore I make the same note as above; although I believe I have been successful in providing *some* triangulation, it does not cover all my data – especially not if we only consider the quantitative back-ups.

In summary, I would say that proving validity is an exercise in thinking critically, as a researcher but *also as a reader*. I have however attempted to provide my part in this cooperation, according to the guidelines offered by Seale (1999)<sup>10</sup>:

“Methodological awareness involves a commitment to showing as much as possible to the audience of research studies... the procedures and evidence that have led to particular conclusions, always open to the possibility that conclusions may need to be revised in the light of new evidence.”

---

<sup>10</sup> Quoted in Silverman (2005)



## Chapter 4

# Empirical findings

---

This section will tell the story of Novell and the openSUSE project, based on the data, conversations and observations I have gathered in this study. These are some of the empirical questions that have been driving this data collection:

- How has Novell handled the transition to Linux?
- How is the development process of the Linux product organized?
- What significance does the openSUSE community have for Novell?
- Who *is* the community?
- What are the strengths and weaknesses of this collaborative model?

Some of the questions will be answered in this chapter, others will be addressed later. Combined with the theory of autopoietic social systems and boundary objects, I hope to show how the case of Novell can give us answers to the research questions stated in the introduction. This chapter is largely written in the words of my informants, with my interpretations of their reality. The theory of boundary objects and autopoietic organizations will therefore be less visible in this chapter, but I will return to discuss the empirical data in relation to these theories in the next chapter. I would also like to note that the majority of the data in this section is acquired in October 2007. The descriptions in this chapter presents a *snapshot* of Novell's evolution. A few months after the data were gathered, the situation had already changed. I will present and discuss this continued evolution at the end of chapter 5.

This chapter is divided in four parts, and will start with an account of **the strategy** behind Novell's move to open source. This section is written in a narrative style with a presentation of

some of the recent history of Novell, and look at some of the reasons for why they chose to pursue the open source model. The second part will detail the open source **software products** that are licensed by Novell. The important point is to uncover how the commercial and the community Linux products relate to each other, and in what ways they are interdependent. Grasping this relationship is important for understanding how **the development process** is organized, which I will detail in the third section. Novell's boundaries have by no means disappeared although they develop open source products, and in this section I will look at the differences between the development process within and external to Novell, and which objects and channels support the production process. External development is largely equivalent with **the openSUSE community**, which will be presented and discussed further in the last section. Who the community consists of is a particularly interesting question, as there are several ways to see this. I will also look at some other variables concerning the community, such as its structure, activity and motivation.

## **The strategy**

It may be useful to start off by repeating my first research question:

### *1. What is the (theoretical) distinction between the sponsor firm and the sponsored community?*

In order to understand where Novell is today and what the openSUSE project is, we need to have a look at the recent history of the company and their move to open source. Novell is an American software company with a long history in the global software market. The company was originally founded in 1983, and introduced the first LAN software based on file-server technology in the early 80's. The operating system, named NetWare, went through several stages of development during the decade and increased in popularity. Ten years later, Novell was a leader on the enterprise market with a 70% market share. Novell's hegemonic position was threatened when Microsoft in the late 90's introduced new network products and a strong competition, reducing Novell's position in the market despite the company's efforts of adapting its products to Internet technology. Although NetWare was a strong product that had held a superior technological position, it became clear that this operating system had reached the end of its life-span, and had a limited future potential. At this time Novell had also developed a range of enterprise software products that were sold in addition to NetWare. One of Novell's strengths was the company's ability to deliver a full suite of software products for enterprise customers, ranging from the operating system to enterprise applications such as email-clients, security and server systems. However, without a strong operating system in the future portfolio, this strength would be gone. According to a consulting manager within Novell, the company therefore needed to find a future solution to this problem:

“Novell had an industry leading technology with NetWare, but the company’s exec’s understood we were facing this gradual decline into obsolescence. The only way for a company our size to avoid it is to “jump S-curves<sup>11</sup>”, meaning to provide a path for loyal customers to move from the current but aging platform to a new upcoming and innovative platform” (interview #1, 18.05.07).

It came as a surprise to many that the strategic choice for Novell would be to use open source technology and Linux to fill the gap from NetWare’s descent. In 2003 Novell acquired two large open source companies (Ximian and SUSE Linux), on which their new strategy would be built: Traditional Novell technology would now be fused with open source products and integrated on the Linux platform. The manager continues:

“Linux is now the foundation for our product strategy. SUSE Linux subscription revenues are intended to eventually replace the declining NetWare license revenues. Linux is our new “S-curve”, that we believe will allow us to avoid the gradual obsolescence and decline that afflicts technology companies. (...) Our challenge, along with all other open source software companies, is to monetize our investments in open source, but I think we have a good strategy to be able to do that” (interview #1, 18.05.07).

I spoke with one of the people that – at the time – participated in pushing Novell towards Linux from the inside (interview #6, 25.05.07). He elaborates on this dire move: “The thinking behind it was: we’ve got to do something to make Novell an interesting company again. Cause it’s not.” However, the transition he was pushing for was not an easy process, as there existed little understanding within the company for what open source really was:

“In the early days there were literally maybe 3 people in the company that had any knowledge of open source, and I’m not exaggerating. (...) the biggest problem was culture, changing the culture around open source: “What do you mean, you’re gonna give the product away?!” And sales guys were saying: “wait a second, I can continue to sell maintenance services on our proprietary products and make a lot more money than I can by selling this cheap little Linux thing”. So you’ve got all sorts of disincentives of going open source with a company that has been born and bread on proprietary software and big fat license revenues upfront” (interview #6, 25.05.07).

The idea was, however, not to create large revenues on licenses for the new operating system itself, but to support other products: “the thought wasn’t really to make money off the operating system, but to hold off the decline of Netware as long as possible, and to start to provide an attractive platform for Groupwise, ZenWorks and all the other traditional Novell products to

---

<sup>11</sup> An S-curve defines the progress of a product based on a particular underlying technology, and can be drawn on a two-dimensional chart with efficiency on the y-axis and time on x-axis. When two or more s-curves for related technologies are plotted on the same axes, any discontinuities between the technologies become visible. A newer technology S-curve may appear to provide greater marketable value than an older one. In such cases, a company may gain competitive advantage (which can increase profits) by adopting the new technology - in other words, by **jumping to the new S-curve**. The rationale for making that leap is strengthened by the possibility that the new technology will ultimately replace the old one.

build on.” (interview #6, 25.05.07). Today, Novell group their services on four areas: (1) Datacenter and desktop solutions (Novell SUSE Linux Enterprise Server), (2) Security and identity management systems, (3) System and Resource Management systems (ZenWorks), and (4) Workgroup solutions (Groupwise). The open source Linux operating system has replaced NetWare as the foundation of Novell’s product portfolio, which other proprietary products rest upon. With the release of SUSE Linux Enterprise 10 (SLE10) in august 2006 under the accompanying slogan “Your Linux is Ready”, Novell delivers Linux solutions for enterprise businesses from servers to desktops. Novell still supports NetWare for its existing customers, but has created solutions for slowly migrating these customers over to the SUSE Linux platform in time.

### **Going for SUSE Linux**

When the choice was made to go for Linux, Novell needed to find a way to get open source development into the company. As Linux is a open source product, Novell could have chosen to build a Linux distribution within the existing company structure. But without engineers with any Linux-experience in the company, it would have been a daunting task to build up such a department. Instead Novell looked at the possibilities of acquiring a company surrounding an existing Linux distribution. SUSE Linux was a German company that had been developing a commercial Linux distribution since 1996, and had recently started targeting the enterprise market. According to informants I spoke with at Novell’s Germany office in Nuremberg (formerly SUSE), SUSE had been struggling financially following the dot-com crash and was seeking venture capital. Novell’s purchase of SUSE Linux in 2003 therefore seemed to solve the needs for both companies. Novell thus adopted the SUSE Linux distribution, branding it as a Novell product while keeping the engineers in the German company onboard to continue the development of the distribution.

Novell’s first launch of Linux was branded as “Novell Linux Desktop 9”, and was marketed across the world in Novell’s red company colors. According to former SUSE employees, it was a mistake to drop the pre-existing SUSE brand name in the first release: “It was red, all the branding was gone, and we were furious! (...) But then at some point somebody had noticed that SUSE Linux was a very very strong brand.” (interview #23) According to this engineer from the former SUSE company, it seemed as if Novell had underestimated the strength of the SUSE name among the existing customer base and open source community, and therefore chose to keep the SUSE name in its future Linux releases. The next (and current) enterprise release was named “SUSE Linux Enterprise”.

The SUSE Linux company had traditionally focused on producing a Linux “consumer box” that was available for sale in retail stores. The sale of this box accounted for approximately half of the

---



revenue of the company, and was primarily sold in Europe. SUSE's ability to sell an open source product was largely due to the closed-source development process within the company. Although SUSE Linux had always been open product licensed with the GPL (as all Linux distributions are required to), it was not possible to retrieve the source code for the next release until it was ready for purchase. SUSE's strategy was to create a technically superior Linux distribution with the large number of employed engineers, that would make users willing to pay for their distribution in retail stores. One of the German managers in Novell recalls how the development process worked in the days before Novell:

“What we did was we collected open source software from the community and numberless projects, and then we moved the software inside the company and we closed the door again. Development from that point, the development of the distribution, was done largely behind closed doors” (interview #8).

This consumer box, produced behind closed doors within SUSE, primarily targeted home users and small businesses in Europe. Novell, on the other hand, never had much interest in home users and held its strongest position in the United States. After Novell acquired SUSE, emphasis was therefore put on developing Linux for the enterprise level and configuring the distribution to be compatible with Novell's existing enterprise applications suite. The consumer box was still sold the first couple of years since the acquisition, and provided some revenue for Novell. The role of the SUSE consumer box was however about to change.

### **Birth of the openSUSE project**

In April 2005, a secret project started to unfold within Novell's Linux R&D in Germany. Following an initiative from the Novell headquarters, a team of five started planning the steps of opening up the development process of the SUSE Linux distribution. Novell was aiming at converting the SUSE Linux distribution to a community-based product, similar to other open Linux distributions such as Fedora, Ubuntu and Debian, to name a few. This would mean an end to the closed development process in the SUSE department, and allow outside developers to influence the distribution. The “openSUSE project” was announced four months later, in August, “with a clear goal to give something back to the community, to make it possible to contribute and to participate.” (interview #8). Before this, SUSE had nothing but a user-community and a group of invited beta-testers (approximately 200 people), who had a marginal influence on the product. The openSUSE project was an attempt by Novell to create an open community surrounding the SUSE Linux distribution, and attract users and developers that would collaborate in developing and spreading the openSUSE Linux distribution. Novell's official relationship to the community is being its “founder”. An open source marketing manager in Novell claims that the openSUSE community is quite autonomous from the company: “So

what we did is that we created the openSUSE project, which is owned by the community. We don't own it. We provide some funding and supply some hardware. So they kind of take the project in the direction they want to." (interview #5, 25.05.07). We will return to discuss the community's independence from an empirical perspective later in this chapter.

The establishment of the openSUSE project required a change in the organization and development process within the SUSE department, which was not met without friction and resistance. As the news of the project spread through the SUSE department, the internal mailing lists were filled with discussions. The decision to open up the distribution created some disturbance and concerns among the German engineers as the consumer box had always had a pivotal position in SUSE, delivering important income for the survival of the company. By creating openSUSE, Novell would offer the Linux distribution for free download to all users, and sales of the consumer box would obviously drop. This was not a big concern for Novell, whose business strategy did not rely on the income from sales of the operating system to home users and small business. But the project also caused concern for some of the employed developers for other reasons:

"The original push came [for the openSUSE project] from very very high on top. And it was against resistance here, because - there were some people who are now the main people in the opensuse project who saw a chance [for it to work] - but there were many people who were very strongly against it. I mean, it will change the way we work, and it has to be done with a little bit of care to make sure that not only what the community needs is looked at, but also what the developers here need" (interview #23)

One of the top managers in the SUSE department has also experienced that the new project has created some anxiety among employees, although he believes this is uncalled for:

"I encourage people [employed developers] to be more outside, I encourage them to communicate upstream and also on the opensuse mailing lists, to actively participate. But for some of them it takes time. (...) And sometimes [they] also fear that: 'if I am working with somebody outside, that person can do my job. Will I not obsolete myself?' That is a fear that I do not see there, because I only see that it allows people to focus more on quality instead of quantity. But, yes, that fear is there" (interview #14).

Why did Novell establish the openSUSE project? Based on the previously reviewed literature, there could be several reasons for a company to facilitate an open source community. The reasons may include wanting assistance with development of the product, testing of the software, marketing and diffusion of the product by establishment of a large user base, assistance with translating the software to other languages and establishment of ties to open source developers for recruitment purposes. The openSUSE project has now existed for more than two years, and several of these benefits are confirmed to be present by the informants I have been in touch with. For development assistance, a packager within Novell says: "So yes, we need the help, and we

need the community [...] Because we are just overwhelmed because we are not enough people to do the same work as Debian does. Yeah, it sounds crazy with 200 full-time employees, but it is a lot of work.” (interview #23) A development manager responsible for the enterprise Linux distribution also claims that the community gives valuable assistance with testing of the software:

“We of course benefit from the openSUSE community. First of all, we’re trying to always have an openSUSE release prior to the Enterprise release, using the openSUSE as a test vehicle for newly introduced technologies. *The best test coverage you can get is from the community*, this means we’re trying to implement possible new technologies for the enterprise version first of all in the openSUSE versions” (interview #26 – my emphasis).

The relationship between the software products mentioned in this quote will be closer detailed in the next section of this chapter. In addition to assistance with development and testing, the openSUSE community has proven valuable on other areas as well. An employee with a special responsibility for communication with the community confirms that the community contributes with translation of the software to languages that are not supported by the Novell company:

[Myself]: “So the translation performed by the community becomes a key for Novell to access markets outside the [supported] languages?”

[Employee]: “For sure, yes. Especially for smaller languages. Like Vietnam, for example, would not have had a distribution if the community didn’t do it” (interview #10).

Having ties to the community outside of Novell has also paid off for the company in terms of reaching out to future employees, as this team manager in Novell confirms:

[Myself]: “Has ever SUSE or Novell employed anyone from the community?”

[Manager]: “Yes. I mean, all developers here, that are not from way back, were in the open source community before they got hired by SUSE or Novell” (interview #9).

Unfortunately it is hard, if not impossible, to measure the impact of these benefits in quantitative terms, and thus find out *how important* they are for the company. It may also be that some of these benefits are merely fortunate, unintended side-effects of the project. Nevertheless, we will go into more detail about the relationship between the community and the company in the next sections, and hopefully uncover more about the importance and consequences of this relationship. But no matter the impact of the mentioned benefits, there may also be other strategic reasons for Novell to launch the openSUSE community project. One of the managers speaks his personal opinion about how he believes that the move was more about competitive positioning:

“This is my private opinion, at that time Novell was basically forced to do something into this kind of community-driven distribution thing, because Red Hat had just launched Fedora a couple of months earlier. So they had to do something publicly, so they basically started something to compete with Fedora” (interview #11).

Another strategic motive may be that Novell needs to build credibility and legitimacy to operate within the realm of open source. To borrow the terminology from Bonaccorsi et.al (2007), it may be important to be perceived as a *giver* and not only a *taker* to achieve legitimacy in the open source domain. According to an openSUSE manager, the openSUSE project is considered as a “gift” to the larger open source community by Novell (interview #10), since the previously closed development process has to a larger degree become more transparent and opened up for everyone. It is apparent that Novell intend on contributing back to the open source community in this fashion, as they also released the source and opened up the development process of the “AppArmor” software after they acquired the owning company in 2004. For a company with a long history as a proprietary software company, it may be especially challenging to become a legitimate actor in open source development - as the former Novell-strategist points out:

“The first big challenge - and this is one that Novell really struggles with – is feeling like you are in fact a community. And the more corporate the intention behind the community, the less community there will be. (...) Novell has more of an image problem than most of the [open source] companies do, because of the Microsoft patent deal, because it’s a latecomer to open source and because it has this hybrid model” (interview #6).

## **The result**

What does the openSUSE project consist of? In general terms, an open source community needs some structures and tools to facilitate the collaboration, in addition to the people that are involved in the work. In her study of firm-sponsored communities, O’Mahony found that this is also true for these kind of communities.

“In forming an open source community, sponsors borrowed heavily from the tools and techniques pioneered by autonomous open source communities. This included general-purpose online tools such as web pages and particularly e-mail discussion lists. However, most also used some of the tools specifically developed and refined for open source software production, notably the CVS source code control system and the Bugzilla error tracking database” (O’Mahony, 2007a).

Building on O’Mahony’s work and the discussion on open source communities and methodologies in the previous chapter, I propose a summary of some of the most important elements of an open source community (firm-sponsored or not). In general, an open source community should include:

- (1) The source code and binaries to the software product
- (2) A collaborative information platform
- (3) Communication channels
- (4) Development tools
- (5) A governance structure

(6) Collaboration protocols

(7) A unifying identity

For openSUSE, it may seem as if the community has matured in only a few of these areas. At the time when openSUSE was initially launched, the project was released with continuous access to the source code of the distribution (1), a project wiki (a website editable by anyone) serving as the collaborative platform (2), open mailing lists and IRC channels (3) and an open bug tracker system (Bugzilla) serving as the main developer tool for interaction between the employed engineers and the community (4). As for governance structures (5), Novell initially held full control of all decisions in the project, and gave little authority to the community itself. Protocols of collaboration (6) were initially quite unclear on how external contributors were to interact with Novell. (This description of the early situation is paraphrased from the interview with a top manager of openSUSE - interview #8). The last element on the list – a unifying identity (7) – is an aspect I will return to in the next chapter, as I find this element to be largely ignored in much of the open source literature. At the launch of the openSUSE project, this identity was everything but clear, as this new distribution drew upon the identity of two previous proprietary/commercial companies (Novell and SUSE) moving into a new space with the openSUSE distribution, trying to shape a new identity. We will see that this situation is different today.

Since the launch, more elements have fallen into place in the community structure. The collaborative platform and the communication channels are still the same, but have grown considerably in its size and use. A major development tool has later been introduced - the openSUSE Build Service – which allows any external developer to modify, collaborate on or create their own software projects. The Build Service is a strong innovation for the community, facilitating external software development to the distribution (this will be described in more detail later). Another recent development tool that has been introduced to the community is a feature request tool (FATE - introduced November 2007). As for development protocols the project has recently established the Guiding principles, created by Novell in collaboration with the community. These outline the relationship and nature between Novell and the community. Novell has also transferred some limited decision-making power to the community through the establishment of the openSUSE community board (November 2007), and is also working on the early development of a trust-rating system for authorizing external developers to influence the distribution directly. The governance structures of the openSUSE community are however not at a level comparable with the principles offered by O'Mahony for community-managed projects (O'Mahony, 2007b), and neither the community facilitators within the Novell's Linux R&D nor the community members are completely satisfied with the current configuration of the community structure (personal observation). For instance, there is today still no opportunity for external community developers to contribute to the code of the distribution directly. As will be

---

shown, any contributions are currently implemented at the mercy of Novell engineers and product management. From conversations with external community contributors, this is an aspect that is sorely missed in the community. We will return to discuss the participation of external developers later, after we have had a look at the specific open source software products that are licensed by Novell.

## ***The software product(s)***

If you are interested in getting a Linux operating system from Novell, you basically have two options. If you are an enterprise customer, you would most likely be interested in purchasing licenses for the SUSE Linux Enterprise (SLE), which may include support, training and updates for the system. You will also be purchasing a system that has been thoroughly tested and optimized for an enterprise environment. The other option is to download the openSUSE distribution, which comes free of charge. This is an open distribution, meaning that its development has been influenced by external contributors from the outside community. When using this distribution free of charge you are basically on your own - if you want support and help on how to use it, you need to seek it yourself, and the distribution comes without any guarantees from Novell that it will work on your computer system. However, the openSUSE distribution has an open community surrounding it, meaning that if you sign up on mailing lists or join their IRC channels you are likely to get assistance from peers there. These are the basic differences between the two alternative Linux directions offered by Novell, seen from the user's point of view. In order to understand Novell's strategy and development model, we nevertheless need to detail the differences and the relationship between the open community distribution and the enterprise distribution closer.

### **Number of users**

How large is the user base of openSUSE compared to the enterprise products? The user-ratio may impact the perceived importance of the distributions in relation to each other, and is therefore interesting to find out. It would be reasonable to expect that openSUSE is used more widely, as it can be downloaded free of charge and requires no registration. On the other hand, this distribution has only existed a couple of years. In comparison, the enterprise distribution has been around twice as long and has received all the marketing recourses by Novell. I did not find any official numbers on the user-ratio between the two product groups. To compare the user bases of the respective products I therefore chose to study the number of downloads of comparable products within a given time frame, with the help from a Novell engineer<sup>12</sup>. **I found that the ratio between enterprise users and openSUSE users is roughly 1:5, meaning there are five times as**

---

<sup>12</sup> Thanks to Jürgen Weigert for providing the help.

many users of the openSUSE distribution compared to the enterprise release. The numbers have been calculated as follows: For openSUSE, I have counted the downloads of the DVD version of the 10.2 release. This product accounts for 67% of the openSUSE downloads (other downloads include ftp-versions, beta versions and more). It is reasonable to expect that approximately all users have downloaded the DVD version once, and the other versions account for multiple downloads by the same user (often made available before the DVD version). There is still a risk that many users may have downloaded the DVD version multiple times, and making the number of openSUSE users artificially high. The numbers should therefore be used carefully. For the enterprise products, I have counted the total downloads of SLE Server 10, SLE Desktop 10 and the Service pack in the same time period (the Service Pack was released some months later than the original release and included the original and full distribution, so that new users would simply choose this download option). Since downloads of the enterprise products are limited by licensing costs, there is less of a chance that there are multiple downloads by the same user. On the other hand, there is a risk that users downloading the Service Pack have previously downloaded SLES/SLED 10, but I believe this problem is roughly outweighed by the similar risk with the openSUSE download. Since openSUSE and SLE follow different release cycles, I counted the number of downloads in the period from January to September 2007, to include the initial and later release phases in both products. In total there were 583310 downloads of openSUSE 10.2 compared to 139169 downloads of enterprise version in this period.

### **openSUSE vs SUSE Linux Enterprise**

What is the difference between the enterprise and the openSUSE distribution? Technically, the enterprise and the open distribution are very tightly related. In fact, one might say that they are basically the same product with emphasis on different technical aspects. All distributions released by Novell (openSUSE or SLE) come from the same pool of software source code. This pool is called the “common code base” or the “openSUSE code base” (hereafter referred to as “the code base”), and includes all the different versions of packages and source code for all the different distributions and releases. The code base consists of software that for the most part is collected from the larger open source Linux community (upstream), with adaptations and enhancements added by Novell engineers. Novell also develops some packages/modules that are more or less exclusive to SLE and openSUSE, such as the YaST setup tool.

Since Linux development happens constantly over the entire globe and among a vast amount of individuals and companies, the openSUSE code base is also under constant evolution and development as updates are introduced on a daily basis. Approximately every 8 months a new openSUSE Linux distribution is released. (The release cycle is displayed in figure 6 below.) In some ways a release is a snapshot of the latest developments in the code base, although it is a fairly

planned process. Most packages that exist in the code base are included in this openSUSE release, with the exception of packages that are only needed for enterprise use.

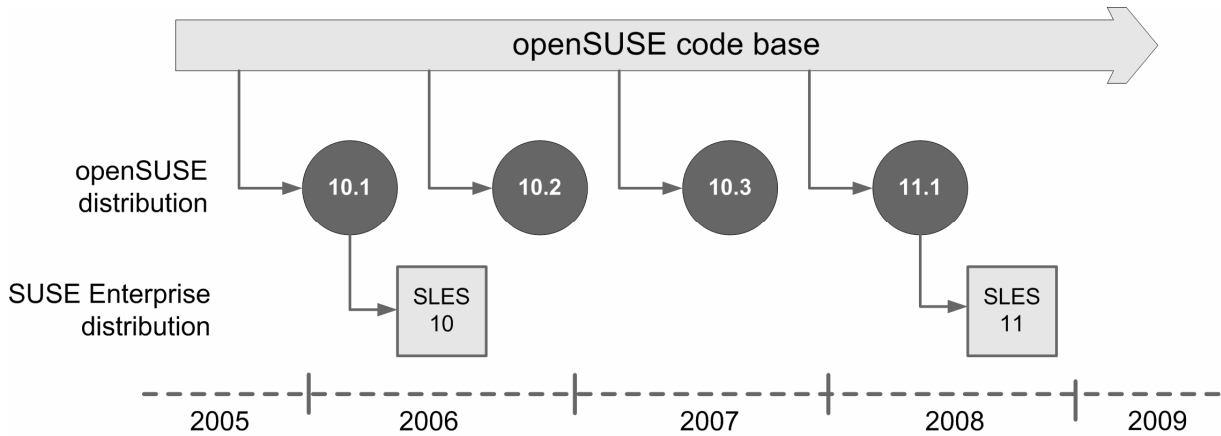


Figure 6. *openSUSE release cycle (dates not exact).*

The enterprise version is built in somewhat the same fashion as the openSUSE distribution, but there are a couple of differences: Instead of cooking this release together using the latest developments in the code base, the enterprise distribution is based on *an existing openSUSE release*. This is also illustrated in figure 6. A new enterprise distribution is created for every third openSUSE release, approximately every 2 years (20-24 months). Since the openSUSE release has already undergone extensive testing, it serves well to use it as the base for the enterprise release which holds even higher quality standards. Many of the packages from the openSUSE version are dropped in the enterprise edition, as the Novell-customers do not need most of them. According to an openSUSE manager, an average openSUSE release includes approximately 3500 packages, while the enterprise release only includes roughly 1500 (interview #8). The packages that are included in the enterprise release are then subjected for even more testing and improvements. In addition, some packages are included that do not exist in the openSUSE version (these are enterprise packages that few openSUSE users have an interest in using). A Linux-marketing manager in Novell describes the process of adding extra features to the enterprise distribution as follows:

“The challenge is to manage your enterprise product line, and to try to fill in the gaps between what the community wants and what the customer wants. (...) We then, about every 18-24 months, roughly, take the best of the best from openSUSE and we also figure out what an enterprise customer wants. So let’s say an enterprise-customer wants features 1 through 100. openSUSE might have features 1 through 93. We’ll take our engineers and have them write those additional 7 features those customers wants. It will all be open source software, it will all



be stuff that we contribute back to the community. But these are SUSE engineers who are on our payroll, so we can dictate a little bit what they write if they want to continue to get a paycheck. It will be 100% open source, we will never own anything, but the gap between what customers want and what the community wants is somewhat different” (interview #5, 25.05.07).

As shown, the openSUSE and SLE distributions share the same code base. The important implication of this fact is that *changes made to a package in the openSUSE distribution will most likely reverberate into the enterprise distribution as well* (given that the package in question is included in the enterprise version). This means that when an employee is working on improvements to the openSUSE distribution, the work will at the same time benefit the enterprise version as well - or vice versa. This is why Novell’s Linux R&D does not distinguish between employees working on enterprise products and the openSUSE distribution, as this manager illustrates:

“All the different departments have stakes in the common code base. Even though they are different departments, they are all working on one central product. And they can have their own products that are based on the common code base. And if they want to change a package, they basically change it for all the other projects as well. So we really try to stick to a common set of packages, which makes maintenance and all the service offerings easier, and keeps the synergies up. So people working on the enterprise are also working on openSUSE, and vice versa. It can depend on what time of year we are in. So we can have a developer that is working a month on openSUSE, and in parallel is working on a service pack for SLE. And then a month later he is working on the new SLE and maybe another project. So basically all the people are working on many different projects at the time” (interview #11).

As the two Linux products are so tightly coupled, it is interesting to ask how the interests of the two distributions are aligned. When there is a conflict of interest concerning the enterprise version versus the openSUSE distribution, which takes priority? Novell does not hide the fact that the enterprise and customer interests must be considered first, as stated here by an openSUSE engineer: “Enterprise takes priority, for sure. Because Novell pays the developers, and they need something back.” (interview #10). Among the community contributors I have been in touch with, this arrangement is widely understood and respected (interviews #17, 18, 21, 24, 25). However, since the products are very tightly coupled, remarkably few conflicts arise. They basically share the same interest: “If you make a kick-ass free distribution, the enterprise distribution is also good.” (interview #10). One of the openSUSE managers claims that the mutual interest of the two products makes it difficult to put one in front of the other: “You can’t compare that, it’s an ongoing give and take. (...) It goes both ways. This also a little bit different from Red Hat, because they are more decoupled than we are.” (interview #14). The tight relation between the two distributions is also embedded into how the development department is organized, as most developers work simultaneously on both products: “There is no one specifically assigned to a product (...) most of the people here work on every product. For

instance my team works on openSUSE, on the enterprise desktop, the enterprise server... those are the same people.” (interview #11). This manager continues to explain how this helps the developers understand the different concerns when he says that “...the employees feel the pain of all the involved parties because they are working on all products and all these different groups.” (ibid). However, problems may arise:

[Myself]: “Does it ever get chaotic, or does it always run smoothly?”

[Manager]: “Depends [laughs]. As soon as one party of interests gets too much power, then it gets chaotic. Because then you have to force their decisions onto other groups, and then it usually gets chaotic. But it is not so often the case. (...) For instance, two versions ago, the enterprise distribution had some requirements that were platform-wide. They said they wanted to have their own package-management stuff, and then all the other groups were forced to do it their way, and it was totally not aligned to the requirements of all the other groups. So that one got really messy” (interview #11).

The few problems that *do* arise are mainly related to the fact that the users of the products are very different, and may have diverging interests. A developer working with storage and file-systems explains how this can create tricky situations:

“Currently we have the same code base that is used for large server machines and for laptop of a single user. And especially in the storage part it is not easy to handle with the same code base, because the target audience is quite different. You can get complaints from companies saying: ‘if you have more than 500 disks attached, it’s too slow’ [laughs]. And at the same time you get requests: ‘yeah, make it more comfortable, you should detect much more stuff and look at every file system on there’, which would make it even more slow for the other use-case. Sometimes these are simply contradicting each other” (interview #19).

## ***The development process***

We have had a look at the rough differences between Novell’s two GNU/Linux distributions, and how they are related. This section will describe closer how the software is developed and show how the Novell engineers and external community developers collaborate in the software production. It can therefore serve to repeat my second research question:

*2. What is the nature of the relationship between the organizational and the external interactive system, and how are the two systems bound together?*

To start off, we will look at all the groups that are involved with the development of Novell’s Linux software. When we thereafter look at how the development actually is done, we will encounter the boundary objects that provide parts of the answer to the second question above.

**Who are the developers?**

It would actually be impossible to count all the hands and minds that are somehow involved in the development of any openSUSE software release, since most of the packages included derive from the larger GNU/Linux community. This community consists of millions of individuals and thousands of companies all around the globe who have participated to a larger or smaller degree in creating and developing the Linux software. In the development process within Novell, the company continuously interacts with this community by gathering software and contributing improvements to the same software back to the upstream community. If we hypothetically imagine an openSUSE release to have a finite start and end, the development process would begin with gathering the main body of software from the Linux community in the first stage of development. The selection of which software packages to include is actually part of what creates the unique identity of the openSUSE distribution.

In the second stage, the gathered software is further developed by Novell engineers for the purpose of adapting and adjusting it to the openSUSE/SLE distribution and the needs and requirements of the openSUSE/SLE users. The distribution consists of a few thousand packages, whereas some are more important than others. The effort that is put into refining these packages is dealt with accordingly. Some packages are merely adapted so that they fit and work along with the rest of the software in the distribution, whilst others are assigned a lot of development resources. For example, there are currently two engineers that are involved with maintaining the Apache software package for Novell. Apache is the widest used web server software in the world, and is a large piece of software. Needless to say, these two engineers within Novell will have their hands full on just keeping track of the latest developments, testing the new software from the Apache community and integrating it with openSUSE. Once in a while they may perhaps discover a bug that they are able to fix, and send a patch back to the upstream Apache community. This will however be the limit of Novell's contribution in this case. On other software packages, such as OpenOffice, KDE and the Linux Kernel, Novell employs large teams and groups of developers that contribute extensively to the development of these software packages. There are also a few packages that are unique for the openSUSE distribution, such as the YaST2 administration tool. These have been solely created within Novell (former SUSE), and were previously developed in a proprietary manner until the openSUSE project was established. They have now become packages that are available in the larger Linux community, where Novell serves as the primary upstream maintainer.

The development of openSUSE and SLE is done within Novell's Linux R&D. The main body of this department is situated in Nuremberg, Germany, where approximately 300 engineers are located. In addition, the department has offices in the Czech republic, India and in Boston, USA, where additional engineers contribute. Surrounding the development in Novell, is the openSUSE

---

community, where individuals around the world may be involved with the openSUSE development process. The level of contribution to the software product by the openSUSE community is rather moderate, compared to the amount of work that is done by employed engineers. One of the managers within Novell guesstimates that approximately 10-12% of the code that is developed in the second stage is contributed by the openSUSE community (interview #9). According to my informants and personal observations, the external openSUSE community contributes in a largely *reactive* manor, by testing, reporting bugs, and perhaps contributing with fixes to an existing openSUSE release. Nevertheless, there are also examples where non-employed community members play a *proactive* role in initiating, coordinating and executing development on certain packages and areas. I will soon return to how collaboration with this community is facilitated by Novell, which is crucial for understanding their level of involvement.

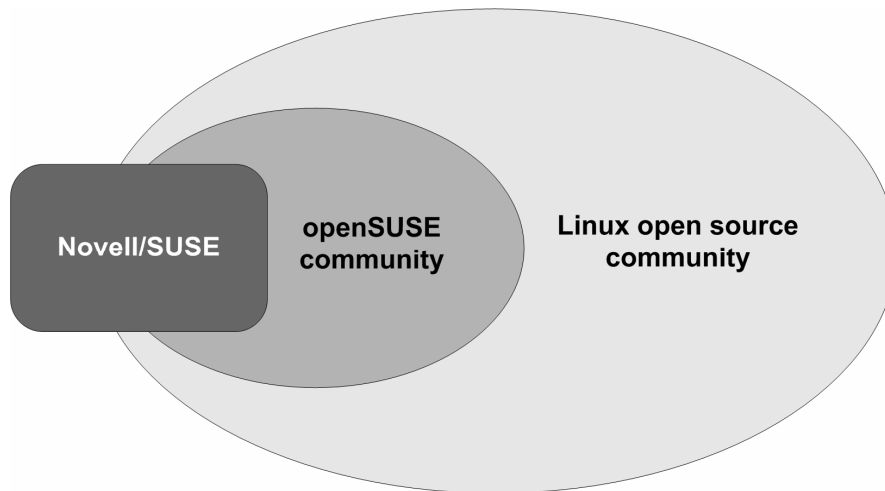


Figure 7. *Communities within communities*

To summarize, figure 7 illustrates the general groups involved in the development of the openSUSE distribution. This model simplifies the affairs, as separating these groups may sometimes seem artificial. We will see how Luhmann's theory of autopoietic social systems can make sense of the distinction between these groups in the next chapter, and answer our first research question. As the figure illustrates, each group belongs to a larger community. The dark Novell group consists of the engineers employed by Novell that work on the openSUSE distribution. According to one of the managers, all employees are encouraged to interact with the outside community in their work (interview #14) - and many of them do. Novell employees that work on openSUSE can therefore be considered participants in the larger openSUSE community. Most employees also relate directly to the upstream development in their area of work, and are therefore also directly linked to the larger GNU/Linux community. This manager illustrates this well:

[Myself]: “So as I understand, almost all employees here can be involved with the [Linux] community in some way?”

[Manager]: “Yes, sure, that’s correct. I mean, we are all members of different communities. Because what we do is that we integrate open source packages. *So every project that we integrate basically has its own community.* Unless you just take the code and drop it in, you are involved in the community. If I fix a bug in a package that I maintain – I happen to maintain a couple of packages, because I was involved in the communities before I actually joined SUSE – and whenever I fix a problem in those packages, I do not only fix in SUSE, but I also contribute the package back to the upstream community. And so we are members of those communities” (interview #14 – my emphasis).

### **Internal development and the Autobuild system**

Internally, the engineers are organized in different units and teams within the Novell’s Linux R&D. For the most part these units and teams are formally organized according to what area of the Linux distribution they are working on. For example, there are separate teams working strictly on the Gnome and KDE desktop environments whilst there are other teams dedicated to the Linux kernel. There are also units with generic responsibilities for the entire distribution, such as the Security team and the Quality assurance team, and units that handle the internal operations services and tools. However, during the first days of my visit to the company I had a hard time making the organizational map fit the terrain I observed, based on conversations with employees at the company. Although all engineers have a formal job description and a square box surrounding their team on the organizational chart, in most cases these only account for parts of the work they actually do. Employees are also quite free to join projects and areas of work they find interesting.

“(…) there’s not one brain sitting around trying to figure everything out. And historically, we as a company [SUSE] have this very open way of doing things. Everybody here is entitled to express his opinion, and to influence stuff. And it’s not so much top-to-bottom hierarchy stuff here, it’s more bottom-up, and there are no real boundaries for people here. It’s not like you’re not allowed to work on something” (interview #9).

Another openSUSE manager emphasizes the task-oriented organization of the department: “To do a release, you need to grab everybody. So it’s not: ‘this is my department, I don’t do anything else’, but people should look left and right.” (interview #14). He explains how they put together fluent teams around certain tasks that need to be done, which can be mobilized from all departments. “There are skills where interest is.” (ibid). A good example of how tasks can be distributed across departments is represented in how package maintenance is organized. Although certain engineers have a specific responsibility for maintaining packages, employees all over the company are involved in this work, as the top manager explains: “There are a lot of people doing packages. Nearly everybody here has packages, the developers and most of the managers own

some packages. Even I have a couple of packages! (...) Therefore everybody is somehow involved with the release.” (ibid). One of the engineers remembers how extreme this was in the past:

“Traditionally, everybody at SUSE had packages. That was crazy when I started here, it has normalized a bit now. But every sales-person, consultant, they all had packages! (...) So when you entered SUSE, you actually entered a big green box where everybody built the distribution, and did something else on the side - like marketing [laughing]” (interview #23).

All the internal product development is focused towards the common code base, which serves as a coordinating and mediating object in the collaboration between the individuals and teams within Novell. This is the pool where all the software source code and binaries are gathered. All engineers and teams that work on the software check in their updates to this code base. Since the code-base is internally shared, everyone can check out the status on all areas and keep track of what is going on. Managing the thousands of packages and the multiple distribution releases is obviously a severely challenging technical task. For this reason, SUSE has developed and maintained an internal tool called “Autobuild”, which manages all the code in the base. Autobuild is more than a repository tool that handles version control, it also makes sure that all the software that is checked into the base works together. According to my informants, this is a feature that gives Novell a competitive edge among Linux distributions, and deserves a little bit of explaining: A Linux distribution is comprised of a large amount of software packages, ranging from the deepest kernel programs to print servers, desktop managers and office applications. Many of these software packages communicate with other packages and programs, and are dependent of other parts of the operating system. The relationships between packages that somehow rely on each other are called *dependencies*. A common problem in the domain of the Linux operating system has been handling these dependencies. When one package is updated and needs to be re-compiled, all other dependent packages also need a recompilation to ensure there is no compilation breakage. If this were to be done manually every time an engineer checked in an update in the common code base, it would be a very time-consuming effort for the company. The beauty of the highly advanced Autobuild system is that it automates the build process of the distribution. This means that all dependencies between every package in the distribution are resolved and automatically refreshed when alterations are made to the code.

Furthermore, Autobuild is constructed to support the various phases of the software development process. Therefore, Autobuild is also a term that describes the entire development process itself within the company. One of the engineers responsible for Autobuild gets upset when people merely speak of Autobuild as a fancy repository: “Autobuild is not just a tool, it’s a *process!*” (interview #15). This process has clear defined paths for product development, including steps for committing code, testing, quality assurance and so forth, and covers both technological and organizational processes. Since all engineers work towards the same interface, follow the processes

inscribed and share the access to all areas of software development through the Autobuild system, this system may explain how most of the fluid organization remains coordinated and organized.

### **External development and the OpenSUSE Build Service**

All the Linux software products from Novell are constructed through the internal Autobuild system. The system (technical *and* social) does not grant read- or write-access to anybody external to the development teams within the company. In other words, individuals in the external openSUSE community *have no ability to directly modify or submit changes to the code in the code base*, meaning they have no way of directly contributing to the software development of openSUSE. So what does the ‘open’ in ‘openSUSE’ really mean? The question of what is genuinely required to achieve openness in open source will be left alone for now, but the following shows how Novell facilitate what they describe as openness. We will see that this matches with what O’Mahony describes as *transparency*, and that the case of Novell supports her finding that sponsor-firms lack *accessibility* as a form of openness (O’Mahony, 2007a).

First of all, Novell grants everybody access to download and *read* all the latest developments of the code in the code base. To accomplish this, Novell creates a copy of the entire code base which is readable and accessible from the outside of the company. This copy goes by the name of “Factory”, and is updated on a daily basis. From a user’s perspective, this makes it possible to get hold of and install the latest software that has been developed for the distribution for personal use, even though it might not have been through all the QA phases yet. For an external developer it is also possible to immediately access the source code to the new software, meaning that he can continue the development of the software for personal or other use if he or she wishes. If he discovers a bug or other problem, he can fix it by altering the code (given that he has the knowledge and motivation to do it.) But the improvements or changes he makes can not be submitted back to the common code base or even to Factory. “The development process still is pretty closed, because there is no direct way of contributing anything. You still have to go through some Novell employee.” (interview #9). This fact is one of the weak spots of the current organization of the openSUSE community, according to Novell-employees and external community members alike. One of the previous members of the original openSUSE planning team reflects on this issue:

[Myself]: “How far have you come on achieving your goals with the openSUSE project in general?”

[Engineer]: “Maybe halfway, maybe a little less. Because what is still missing is all the infrastructure we need and the policies to include external people in the distribution-building. And what I have seen as a problem (...) is that there is no clear policy and no clear idea on how this will work. Because we want people to contribute, but we don’t know how” (interview #23).

The reason external developers do not have write access to the code base is related to Novell's concern for sufficient quality assurance and control of the development of the code. Since the openSUSE product and the enterprise distribution are directly related, an external developer that could make changes to openSUSE code would possibly also be able to directly affect Novell's sales-product to enterprise customers. Since Novell supply their customers with warranties and support for that product, they do not wish to give externals the ability to do alterations out of their own control. Nevertheless, Novell management has an expressed desire to create an active developer community surrounding openSUSE, that is able to contribute directly to the code. Novell's motivation for this is twofold; free donations of quality improvements to the code by voluntary individuals would obviously be much appreciated, as long as they contribute positively to the development of the product. The second reason is that having an active community in itself may award some benefits to the company, like free marketing and distribution of software leading to accelerated accumulation of a larger user base. Having an "open" product with an active community also provides Novell with more credibility in the open source domain, which is much needed in the competition with other enterprise Linux distributions. Last, but not least, my observations reveal that many engineers and managers within Novell's Linux R&D identify themselves closely with open source communities, many of them having a direct background as dedicated open source developers. Since openness is one of the norms that constitute the backbone of open source development, it is natural that these Novell employees themselves feel an obligation to make the distribution as open as possible.

Although direct contributions to the code can not be made from the external community, there are indirect ways of contributing even today. Obviously, Novell are more than happy to receive constructive contributions, but can not release control without a system of quality assurance. This is at the moment done manually: "[F]ortunately they [external developers] work together with the maintainers we have inside, so if we see that they really make improvements then of course we work with them together to pull it back into factory or stable [the code base]" (interview #10). There is no automated process for this, and it relies on the initiative of the individual community member: "[T]here is no automatic way and no guarantee from our side that we will check the Build Service [where community development happens]. So the user has to be active and say "hey, look here, this is better" in order for us to do something" (interview #10). The current development model is summarized in figure 8 below. The arrows indicate directions of influence. (The red dot represents a Novell employee; the green dot represents a community member. The Build Service is presented in closer detail below.)



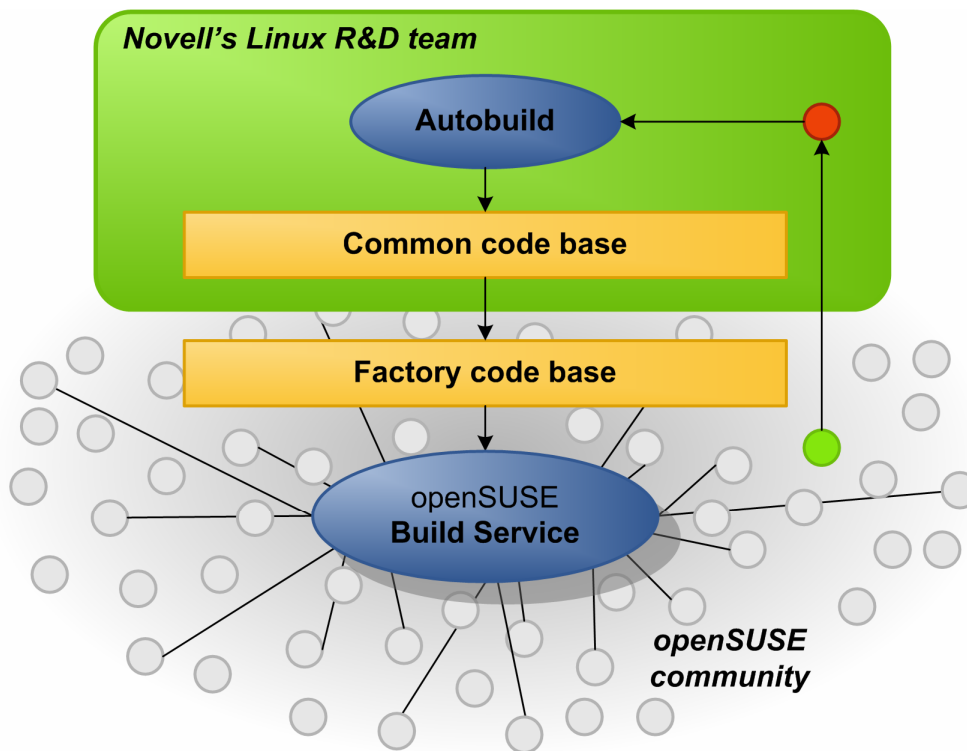


Figure 8. Direction of development - external contributions

After Novell enabled read access to the code base through Factory, their second step in creating an open distribution and facilitating an active developer community was to open up their bug-tracking tool, Bugzilla. This allowed anyone to submit bug-reports, and created some concern in the company, as this manager describes: “And here I remember, prior to the day the decision was made to open up Bugzilla, there was pretty much discussion inside the company if this would be a good idea, as some people were afraid we would be flooded by millions of low-quality bugs.” (interview #8). This fear appeared to be unnecessary. “The number of bugs rose dramatically, but the quality was very good. And it didn’t raise that much that we couldn’t handle the bugs. So this is a very positive step.” (ibid). Today a lot of the collaboration between external community members and employees is facilitated by these bug reports: “If you look at Bugzilla the communication is constant. You can basically go back and forth and ask [the developer]. If you add a comment, everyone else in the bug will see the comment, make replies, so its kind of a constant discussion or communication process that takes place there.” (interview #11). In a few cases the bug-reports may include patches and fixes to the problem that is reported, although this is fairly rare. Bugzilla nevertheless represents an important channel for users and community members to influence the development of the distribution. In the next chapter, I will further discuss in what ways Bugzilla, in addition to the openSUSE Build Service and the Factory code

base, serve as important *boundary objects* in the collaboration between Novell and the openSUSE community.

In parallel with opening Bugzilla, Novell established the openSUSE wiki (a web-page editable by anyone) on [www.opensuse.org](http://www.opensuse.org), to function as the information-centerpiece of the project. Apart from a few official Novell pages, most of the wiki is maintained and authored by members of the community itself, translated to many different languages.

To further support external communication about the development of the distribution, a number of mailing lists were created allowing anyone to take part in discussions concerning the project and development of the distribution. This manager explains:

“From this time on we tried to communicate and interact more and more with the outer community. We created mailing lists, we moved several internal mailing lists to the public, so that discussions that were private to that point, the only internal mails, were then public mailing lists where everybody could see what happens, and where everybody could express his opinions, wishes or so” (interview #8).

The mailing lists are undoubtedly very important channels of collaboration within the openSUSE community and Novell. In addition IRC (internet relay chat) channels are used extensively to facilitate discussion and collaboration. I will return to discuss the role of these communication tools in the next chapter.

Novell’s biggest contribution in creating an active developer community is probably the openSUSE Build Service (OBS). As a part of the openSUSE project, Novell created the Build Service as a service to the community. The OBS provides software developers with a tool to create and release open source software for openSUSE and other Linux distributions. It is accessible to everyone, and any community member can register on the web pages to create their own account. The developer can then start uploading and testing software, and share the software he develops with everyone else. In this way the OBS works as a repository of software, where anybody can access and download software from all the projects that exist on the site. In addition, the OBS comes with tools for managing the software projects, so that multiple people can collaborate on the development. So far, this does not sound very revolutionary as there are multiple sites that offer such infrastructure for open source software development. The unique aspect of the OBS is that it has similar features to the internal Autobuild system, in that it can instantly build your software package for you so it works with all other pieces of software in the distribution you are building it for. It resolves dependencies between packages and can you can even use it to build your own distribution from scratch. This manager explains its function better than I can:

“The Build Service is basically a build system that automates the build process. Its like a revision control system. So that whenever I make changes to a package, the package will be rebuilt, and

all the packages that depend on this package will be rebuilt, just to make sure everything stays consistent” (interview #11).

The OBS can in some ways be understood as a playground for software development, where developers can instantly try out various improvements and configurations on their own and other developers’ software projects. They can also use it to actively work on their own improvements to the openSUSE distribution. For example, a community member can retrieve the source code for the relevant package from Factory, and enter it as a project in the Build Service. Once the improvements are done, he can build his own openSUSE distribution and try it out. The OBS is also being used more and more for building and hosting official software projects for the openSUSE distribution.

The OBS was originally created to give something unique to the openSUSE community, and is the result of an idea that was present from the very beginning of the project. This developer remembers the discussions that went on in the planning-team:

“The real question was: «if we want to be better than [other Linux distributions], what the heck can we do?» And actually, the first idea we had - apart from the obvious «make a community, make a website, make a wiki» - was the Build Service. That was the result of this discussion, because this is something that nobody else had, and that still nobody else has, in that form” (interview #23).

In October 2007, more than 2200 developers were registered in the Build Service. Development work in the Build Service is organized in projects such as KDE, Apache, Beagle, GNOME and OpenOffice.org, and currently includes more than 1200 such projects. According to several active contributors (interviews # 17, 18, 21, 24, 25), the OBS is a much appreciated service in the community. One of the engineers responsible for the OBS in Novell claims that the OBS makes openSUSE the most open Linux distribution of all, in terms of how easy it is to contribute:

“The "open" in openSUSE is very strong in regard of contributing at all. Everybody can immediately contribute in his own project without any approval. He can also join any community project easily by asking the people inside of it. This is more open than any other distribution at the moment” (comment from OBS-maintainer, 08.05.08).

The fact that the OBS allows externals to build and customize their own Linux distribution, partly solves the problem of not being able to influence the official openSUSE distribution: “(...) since we need to keep control of the base distribution for the business products, how can we give the community their distribution too? The idea is then that you can build your own distribution with the Build Service” (interview #23). A feature that makes the OBS especially popular in the community is the fact that OBS can be used to build packages for other Linux distributions, in addition to openSUSE. When speaking with developers at Novell’s R&D team in Nuremberg, I realized that this was a somewhat controversial issue. All the openSUSE developers I was in touch

with appreciated the openness of the Build Service, and my impression was that a few of them spoke carefully about this as if they feared that it somehow would be taken away (personal observation). Again, my valuable informant can illustrate this better:

“We don’t restrict people to one distribution any more. This was not easy internally, because it is not obvious (...) to people who are only interested in SUSE Linux. Because the first thought is that you give up your user-base. You help the other distributions to get more users than you have yourself. But I think it is more the other way around, in the long term. Because, if it works out – which will take a few years – the idea is that people from every distribution will go to openSUSE Buildservice to find their packages” (interview #23).

## ***The openSUSE community***

I was astonished to discover that I received quite different answers to the question of “who is the community?” from different people at Novell and in the openSUSE community. In the next chapter I will show how Luhmann’s theory of autopoietic systems can make this distinction simpler for us, by focusing on communicative events rather than people. However, in order to identify this communication methodologically we actually need to study the people that perform the communications. Let us therefore first look at how employees and community members perceive and frame the community themselves.

### **Who is the community?**

A question that is tightly related to the identity of the community is how big the community is. Novell’s marketing department states that more than 40,000 individuals contribute in the openSUSE community (interview #5, 25.05.07), while some developers at the openSUSE department estimate a size of approximately 200 people. (interview #11). The difference is due to completely different definitions of who counts as members of the openSUSE community. The marketing representative’s numbers are based on how many individuals are registered on the project wiki (the website), while the developer only includes the most active external developers that contribute with a substantial amount of work. I realized that there are several questions to be answered in the attempt of defining the boundaries of the openSUSE community. For example, does the community include employees that are active on community channels? And how much participation does it take to be part of the community? This manager answers by dividing the community into two different parts:

“Some people are just users of openSUSE, you could call them a passive community. Maybe they will sign up for a mailing list and answer a survey. Then you have people that are participating. They are on the mailing lists asking questions and answering questions. Reporting bugs and testing, and may be developers themselves, being in discussions about the direction of the project, writing articles for the wiki, etc. Those people I would call members of the active

community. And that community I would like to grow. And I think you get that community in some ways from the passive community which is much larger” (interview #14).

I encountered the distinction between the active and the passive parts of the community quite often in my conversations with community members and employees. Other names describing the same distinction are the “contributor community” and the “user community”. And as the quote above exemplifies, it is the active part of the community which receives most interest and attention by the openSUSE managers, as they are most important for the development of the distribution. However, not everyone believes it makes sense to differentiate the community in this manner, as this employee illustrates:

“So who is the community? The community are people who are in some way involved in the opensuse project. But we also try not to make such a big difference, because even if you are only a user you are also part of the community. Maybe some people would say “Yeah, he’s a member of the community, but he is not that valuable as other people that really contribute to the project”, but I think this is bullshit. So if you are using it [the openSUSE distribution], if you go to mailing lists or just do some little things you are part of the community” (interview #10).

In order to study the actual collaboration between Novell and the outside world, it is nevertheless of interest to focus on the group of individuals that take an *active part* in the development of the openSUSE project. Of the 40.000 individuals registered on the project wiki, only a slight few of these really interact with the company. So how do we define these active individuals? Do they have to be code developers? Interestingly enough, representatives from the community have created their own official definition for the group of active contributors during the writing of this thesis, in February 2008. The newly appointed openSUSE board wanted to create an identity that would credit these active contributors in the community, by giving them a status as “openSUSE members”. openSUSE members are entitled to official an opensuse.org email addresses and other perks, and is used to officially recognize valuable community members. The board’s intention is to give the member-status to anyone who in some way has put effort into the development, use or distribution of the openSUSE distribution. Here is an extract of their definition, and the criteria to receive such a status:

“openSUSE Members” are specifically distinguished contributors who have brought a continued and substantial contribution to the openSUSE project. They are approved by the openSUSE board.(...) Contributions taken into consideration will include—but are not exclusive to:

- Code and Packaging
  - Wiki Editing
  - Bug reporting and triaging
  - Translation
  - Continued User Support on any Communication Medium
  - Giving openSUSE Talks/Presentation and/or promoting openSUSE”
- (from [www.opensuse.org/members](http://www.opensuse.org/members))

The openSUSE-members definition above does not clearly identify the Novell-employees' relationship to the community. In my own mind I had not included Novell employees in my understanding of the community, prior to my visit at Novell's Linux R&D. However, I was in for a surprise when I initiated the survey that was directed towards "active contributors in the openSUSE community". The survey was conducted with an open invitation to participate, and the invitation was sent to the main developer-mailing lists. Of course, many employees subscribe to these lists as they use them for collaboration with the community. Being aware of this fact, I was still surprised to see that a few of the respondents of the survey turned out to be Novell-employees whom identify themselves as contributors and members of the community. After having conversations with Novell-employees whom interact frequently with the community, it became clear to me that the openSUSE management does not wish to have a clear distinction between who are employed and not, as I have mentioned before. However, this manager explains that it is not easy to erase this distinction internally:

"There's a second big issue we have here, that people here don't understand that we are only one part of the openSUSE community. There is always this "we and them" thinking, like WE and THE community. But we are just a big part of the community, and there is no "we" and "them". Just like people think of the open source community, as if there would be a fixed group of people that have the tag "open source community". It's not like that. I mean, everybody here is also part the open source community, so it's more like, I don't know... this "we" and "them" thinking is probably pretty common for humans..." (interview #9).

This quote also shows that there is not a single, common understanding within the company concerning who the community is, and whether they themselves are part of it. My impression from personal observations is that employees whom communicate actively on the community channels identify themselves as members of the community, while engineers in Novell's Linux R&D whom mostly work in isolation from the external development regard the community as something they themselves are external to. This manager (who is an active participant on community channels himself) explains how it works: "Some of the developers that are working here might not be that involved in this community, but most of the people working here are involved in the opensuse community in one or another way. Like reading mailing lists, being on the IRC, and fixing bugs in Bugzilla." (interview #11). Since the employees who interact with the community are likely to be quite active contributors to the project (they work full-time on the project), they should fit the definition as members of openSUSE. And correct enough, several of the people on the list of "members" are employees within the company. It is also widely acknowledged among external contributors that employees have a legitimate place as community-members:

[Myself]: "So who is the community?"

[External community member]: "It definitely includes everyone that is involved and that is

trying to contribute something. The question for me is rather if one should include people who are just using, and not trying to give something back or not. *But the Novell employees that are either working directly on openSUSE or remotely, for me they are definitely part of the community*” (interview #18 – my emphasis).

### **Community structure**

In the discussion of community-boundaries above, the distinction of the active community is based on the amount and level of contributions by individuals in the community. This group may consist of developers working on improvement to the software product, or other individuals doing translations of text on the web-pages. Somehow they contribute to the purpose of the openSUSE project, other than simply using the product for themselves (as the large remainder of the user-community do). As an informant comments above, it may be artificial to split the community in two and distinguish these two groups (user-community and contributor-community) from one another. However, for the purpose of this thesis it is useful to create a better understanding of the people whom actively collaborate for the benefit of the openSUSE distribution. Therefore I would like to find an operational definition that includes these contributors. It can be difficult to draw the line between the active and the passive community, even if the openSUSE board has provided a definition of what a contribution may be.

Determining what contributions "count" and which do not is not easy. Instead of looking for what contributions various members of the community have offered, it may be better to look for a common denominator for this group. And the one thing they all have in common is their *activity* on various channels of communication. This is where Luhmann's theory may help us in defining the openSUSE contributors, as his understanding of a social systems is not tied to identity, belongingness or action; but to *communicative events*.

As in a true Luhmann understanding, there would be no community without any communication between the users, developers and contributors. In order to do the work they are involved in, they need to communicate with the rest of the community somehow. For this reason, mailing lists and IRC channels have been created and modularized for every area of work within the project. This is where all the communication (communicative events) take place. We could say that these communication channels provide *structure* to the project, as members sign on to the channels in the area where they want to contribute. The communication channels include:

- The opensuse.org wiki
- IRC channels
- Mailing lists
- Informal channels (chat, conferences, blogs, etc)

The **wiki** at [www.opensuse.org](http://www.opensuse.org) is the center-point of all communication in the project, and includes all types of information ranging from news-articles to user-support and technical documentation of the software. The wiki is also editable by everyone, a fact Novell have never had any problems with.

“[T]he wiki is fully open, so anybody could damage something or put illegal content in, but this has never happened. We have had the wiki open now for over two years. So this was pretty successful, this was the start of the project. We opened up the wiki, we opened up bugzilla. From this time on we tried to communicate and interact more and more with the outer community. We created mailing lists, we moved several internal mailing lists to the public, so that discussions that were private to that point, the only internal mails, were then public mailing lists where everybody could see what happens, and where everybody could express his opinions, wishes or so” (interview #8).

While the wiki contains more or less static information and supports asynchronous communication, the **IRC channels** (internet relay-chat) support direct and instant communication between users, developers and contributors. There are various channels for different topics that users can log on to. The IRC channels mainly follow the same structure as the mailing-lists (described below) and therefore include many of the same topics and discussions. The synchronous messaging system makes it possible to hold meetings on IRC, and openSUSE have project-meetings on IRC every second week. Some larger projects such as the GNOME desktop project have regular IRC-meetings as well.

The **mailing lists** are differentiated in various topics and have a various amount of subscribers. The various mailing lists also serve different purposes. In general, there are two groups of mailing lists: those that offer support for the use of openSUSE Linux, and those that are used to discuss technical and non-technical development of the project and the distribution. The largest mailing list is [opensuse@opensuse.org](mailto:opensuse@opensuse.org), where general support for the use of openSUSE is given by other users. For example, if you as a user have a problem with the software (say you cannot get your printer working), you can describe your problem in an email and send it to the list. Other users whom might know how to solve your problem may then respond back to you and help you with troubleshooting. Everyone that is subscribed to the list will be able to follow this conversation and provide any input (if they have any). However, this list is very large (number of posts may exceed 200 each day) and is in English. Therefore multiple support-lists like this one exist in various languages.

Among the development-lists, there are five in particular that facilitate most of the development and collaboration in the openSUSE project. These are:



The **project** mailing list, where non-technical aspects of the openSUSE project in general are discussed. As far as I have observed, this list holds an overview over the entire project and coordinates activities in several areas, leaving the details to the specific lists and channels.

The **factory** mailing list, which is the main channel for discussions on technical contributions to the software distribution itself.

The **buildservice** mailing list, discussing issues related to the use and development of the openSUSE Build Service tool.

The **translation** mailing list, concerning issues related to translation of the software products and documentation to other languages.

The **wiki** mailing list, where all aspects of the information portal are discussed.

### **Community activity**

To find the active external contributors in the openSUSE community it may be enough to group the people whom are active on these channels, as their interaction and communication is the one thing that is common for all of the people that contribute in the project. In the following I have made an attempt to create some statistics on the usage and size of the various mailing lists where active community members are present. The purpose has not been to identify an exact number for the size of the group of active contributors, but rather to look at the level of activity within this group and between different lists. Furthermore, I wish to examine the relative level of activity between employees and non-employees within this group.

In the overview below, I have selected to use statistics from the five developer-lists since these contain the main body of active community contributors in the openSUSE project. In addition, I have included the main mailing list (`opensuse@opensuse.org`), which includes the largest amount of subscribers. This mailing list is mainly concerned with support issues, and may include community members with different levels of involvement, ranging from first-time users to experienced and dedicated developers in the community. I have examined the email archives for these six mailing lists for a period of a year, from September 2006 to October 2007, using the public mailing list archives on `www.opensuse.org`. To help me count the number of messages and users, I wrote a software program<sup>13</sup> that was able to read the name of the sender of every message, and store the amount of messages per user per month per list<sup>14</sup>.

---

<sup>13</sup> The program is written in Java, and is available by contacting the author. Many thanks to Martin Lasarch at Novell for providing valuable help with the Grep command line tool.

<sup>14</sup> Note: The data contains a source of error; problems with the use of different character-sets may cause names that include unusual letters to appear differently, and the software program will not recognize these as the same user in all

| <b>Mailing list name</b> | <b>Registered users 23.10.07</b> | <b>Unique posters</b><br>(entire period) | <b>Number of posts</b><br>(entire period) | <b>Average posts</b><br>(pr month) |
|--------------------------|----------------------------------|------------------------------------------|-------------------------------------------|------------------------------------|
| Project                  | 327                              | 156                                      | 1271                                      | 98                                 |
| Translation              | 179                              | 111                                      | 696                                       | 54                                 |
| Factory                  | 609                              | 388                                      | 6787                                      | 522                                |
| Buildservice             | 358                              | 224                                      | 3243                                      | 250                                |
| Wiki                     | 238                              | 86                                       | 558                                       | 43                                 |
| <b>SUM</b>               | <b>N/A</b>                       | <b>711</b>                               | <b>12555</b>                              | <b>1046</b>                        |

Table 4. Distribution of users and posts on developer mailing-lists.

The overview in table 4 shows that the technical mailing-lists (factory and buildservice) are by far the most active lists. This may indicate that a majority of contributions to the project are technical as well, although this cannot be confirmed simply by comparing the amount of communications on different areas. The column “registered users” is a snapshot of the amount of users that subscribed to the specific list on 23. October 2007. The next column (Unique posters) is the number of users that have sent at least one mail to the mailing-list within the time period (Sep 06 – Oct 07). The difference between the first and second column may be used as a rough guide to count the amount of sleeping users, who only read and never post emails to the lists (the difference will probably be higher as the second column may include users that did not subscribe Oct 07). The row of *sums* shows that there are in total 711 unique posters on all lists for the entire period (a user participating on several lists is only counted once). Below (in table 5) are the similar numbers for the main mailing-list (opensuse@opensuse.org):

| <b>Mailing list name</b> | <b>Registered users 23.10.07</b> | <b>Unique posters</b><br>(entire period) | <b>Number of posts</b><br>(entire period) | <b>Average posts</b><br>(pr month) |
|--------------------------|----------------------------------|------------------------------------------|-------------------------------------------|------------------------------------|
| main                     | N/A                              | 1836                                     | 53998                                     | 2960                               |

Table 5. Main mailing list - opensuse@opensuse.org

These numbers in table 4 and 5 give us an indication of the size and level of activity on the community channels. As the discussion shows above, openSUSE engineers employed by Novell also participate on these channels, and are encouraged to use the mailing-lists as the main channel for communication. What is the relative level of activity between employees and external community members on these lists? As it is possible to identify employees by their email address (@novell.com or @suse.de), we can differentiate employees from external community members.

---

cases. A user may therefore be listed multiple times in the data. After a thorough manual scan of the data, this source of error is estimated to affect 4% of the users, and increase the total amount of users with approximately 7%. The number of posts will however remain accurate.

---

| Mailing list name | All posters | Employees | Number of posts | Posts by employees |
|-------------------|-------------|-----------|-----------------|--------------------|
| Project           | 156         | 36 (23%)  | 1271            | 267 (21%)          |
| Translation       | 111         | 13 (12%)  | 696             | 255 (37%)          |
| Factory           | 388         | 61 (16%)  | 6787            | 1572 (23%)         |
| Buildservice      | 224         | 58 (26 %) | 3243            | 1343 (41%)         |
| Wiki              | 86          | 18 (21%)  | 558             | 160 (29%)          |
| Main              | 1836        | 54 (3%)   | 41443           | 1433 (4%)          |

Table 6. Employee participation on mailing lists

In table 6 we can see that the mailing list that contains the highest amount of activity by employees (both in terms of posts and numbers of users) is the Buildservice and Factory lists. Although there is a large amount of contributions to the main support list by employees (in absolute numbers), these amount to only a very small percentage of the entire activity on this list. Even so, it could be considered admirable that Novell employees do offer support to users on this list, given that user-support is a service customers pay for in the boxed openSUSE version and the SLE distribution. Another interesting aspect in the data, is the relationship between employees participation in numbers of users and posts. Below are two charts comparing the participation between employees and non-employees, in numbers of users and posts. The numbers are taken from the sum of the five developer-lists.

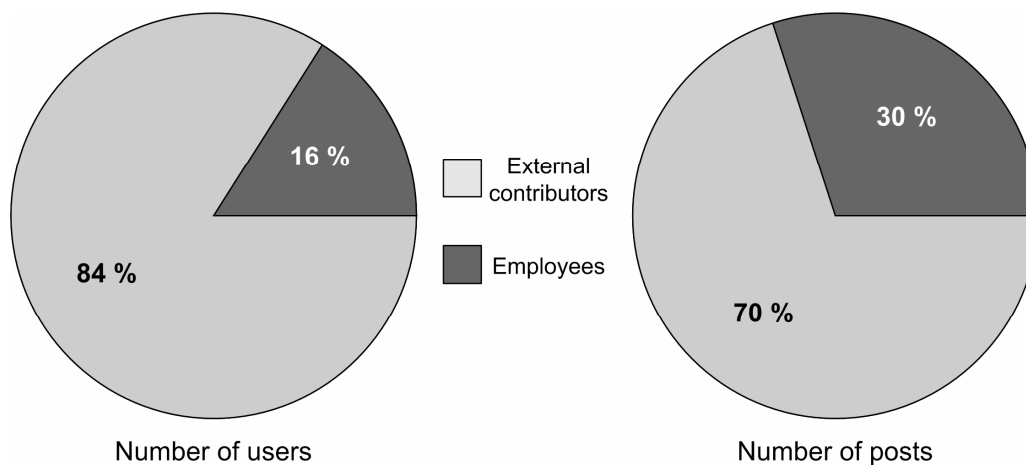


Figure 9. Employee participation on developer lists – users and posts.

Figure 9 shows that although employees only amount to 16% of the amount of users on the developer lists (left chart), they account for 30% of the posts to the lists (right chart). Employees are in other words very active on the community channels, and these data show how the mailing lists serve as an important arena of interaction between Novell and the openSUSE community. In

the following sections we will have a closer look at some of the characteristics of the openSUSE community.

### Community contributions

How much work does in fact the community contribute with? Close to half of the respondents in the survey spend more than 3 hours a week on contributions to openSUSE. 19% contribute with more than 10 hours a week. It is common for open source communities to have a core group that contributes with the largest amount of work. Also in this case, it would seem that the Pareto principle of distribution is a fitting pattern (20% of the community contributes with 80% of the work).

#### 2.1. How many hours in average do you put in contributions each week?

*('contributions' includes time used to prepare input to discussions, testing of software, or other time needed to prepare a statement, user support, code contributions or reports)*

| Less than an hour a week | 1-3 hours a week | 4- 10 hours a week | More than 10 hours a week |
|--------------------------|------------------|--------------------|---------------------------|
| 25 %                     | 33 %             | 23 %               | 19 %                      |

N = 273

#### 2.2. In how many areas do you participate?

*('involvement' means actively pursuing the development of a project by systematically contributing with testing, bug reports, code OR input to discussions. A 'project' also includes non-software work, for example wiki, translations or documentation. If you contribute for the most part with user support, you can answer alternative 2 and provide the amount of time used in the previous question)*

|                                                                                                         |      |
|---------------------------------------------------------------------------------------------------------|------|
| I am only a user, I do not submit contributions                                                         | 18 % |
| I only contribute arbitrarily where I see fit - I don't have a specific interest in any project/package | 41 % |
| I am involved with one specific project/package                                                         | 12 % |
| I am currently involved with one specific project/package, but have been involved with others earlier   | 7 %  |
| I am currently involved with 2-3 projects/packages                                                      | 8    |
| I am currently involved with more than 3 projects/packages                                              | 14 % |

N = 277

*Table 7. Contributions by community*

These results in table 7 may also give us some idea about how much effort is contributed from outside the company. Here is a thought experiment: If we try to calculate how much work the respondents contribute with - using averages for each category ( ½, 2, 7 and 12 hours) - we end up with 1320 hours a week. This is equivalent with the time spent by 33 full-time employees (working 40 hours a week). This is not to say that the community is doing the errand of what 33 extra Novell employees would have been doing, but certainly emphasizes the amount of work

that is voluntarily contributed by the community – and this is only among the respondents in the survey<sup>15</sup>.

The second question in this section shows that contributors are quite spread in the amount of specific projects they are involved in. Members representing the largest category are not tied to any specific project(s). However, statistical analysis shows that the two questions in this section are positively correlated ( $\text{Gamma} = 0,57$ ). This means that the amount of hours contributed increases with the amount of specific projects/packages a member is involved in, and vice versa. It also means that people who contribute less are not tied to any specific projects. This poses an interesting question: Do they contribute less *because* they are not committed to any specific project, or is the lack of specific involvement due to the low amount of engagement? Unfortunately this data can not explain the causal direction between these two aspects. Between these two, I would expect that the latter explanation is more plausible. But, if the former were to be true, it would mean that a larger amount of the community might contribute more if they were given ties to specific projects. Of course, there is most certainly a large amount of completely different reasons that explain the amount of time a member is willing to contribute, so this analysis should not be taken too far.

### **Community motivation**

There are many factors that may motivate individuals to participate and take part of a community, ranging from egoistic to altruistic, and intrinsic to extrinsic reasons. Many of these are covered in the literature on open source, as described in the second chapter. A couple of additional issues were brought up by openSUSE managers I spoke with at Novell, like the value of “getting your footprint” in the product:

“I guess the main factor why open source exists is to see that YOU did something that is useful to others. It is not so much about getting personal recognition or something like that, but like going to the supermarket and seeing stuff that you did [on sale] there. That’s pretty much enough for open source contributors. It’s not like people have to come to you and say thank you, but it’s more like you see your footprint on something, and say “ok, that’s mine. I did that” (interview #9).

Before I arrived at the Linux R&D organization in Novell, I was under the assumption that community members would resent the fact that commercial companies can monetize the voluntary effort that is put into community products. My informant does not agree:

---

<sup>15</sup> Note that some of the respondents in this survey are in fact employed by Novell (as mentioned previously). There are four answers in the open question on “other reasons for participating in the community” that confirm this. How many employees that have replied in total is uncertain, but looking at the remainder of answers I do not believe this is a large group. But it should be taken into account when calculating hours voluntarily contributed.

“I would think the opposite, that people are kind of proud that the openSUSE product is in the smack middle, and not the enterprise products (...) Big companies use that stuff that YOU influence. Your footprint goes into the big enterprises as well. So there is not this sense of that they [Novell] are “using” the openSUSE community to do some work. For the community it [the commercial products] is just a bi-product that Novell sells” (interview #9).

The questions in the survey concerning community motivation are inspired by the conversation above, but also include issues identified in the research on open source communities. The results are presented below, in figure 10.

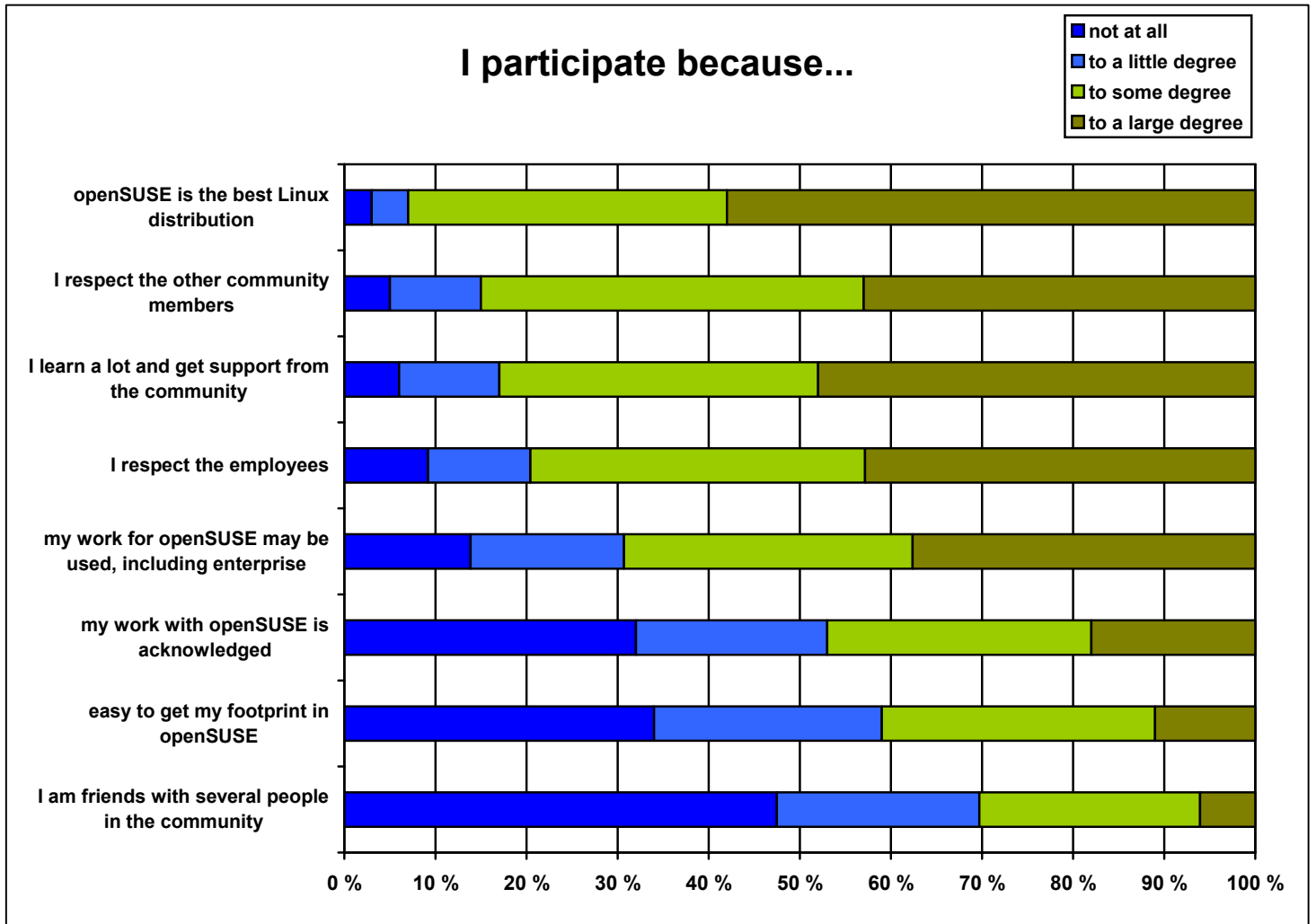


Figure 10. Motivation for community participation - distribution of frequencies.

From the distribution of frequencies in this chart, it is clear that the *quality of the product itself* is the strongest motivating factor, uniting the community. *Personal learning* also matters a lot. Although respect for other community members and employees seems to be important, I would be careful with putting too much weight on these two aspects. The reason is that these questions

might be more sensitive and difficult to answer negatively to, as they concern respondents' view of other people.

It is interesting to observe that 'social ties' is the least motivating factor in the list. There might be several reasons for this, and it is hard to conclude what is the right interpretation. Since humans are genuinely social beings, it is difficult to not be motivated by people that we like. One interpretation of the replies to this question could therefore be that members are not motivated by friends due to the fact that there *doesn't exist* that many strong social ties in the community. This may be natural as open source communities are distributed networks largely mediated through the internet, making it more difficult to establish social relationships. Another interpretation of the low score on social ties may be that contributors underestimate or understate this value themselves; it does not necessarily mean that it is lacking. The initial reason for someone to join a technical community is unlikely to be socially motivated. While social relationships emerge during participation in the community over time, these relationships might not be acknowledged as important motivating factors although they indeed may be important in holding the community together. 30% of the respondents above *do* in fact claim to have friends in the community, and this corresponds well to the percentage of people who contribute with a large amount of work in the community (question 2.1). With a Gamma equal to 0,4, statistical analysis confirms that there is a correlation between amount of work contributed and motivation by friends; the more you contribute, the more likely you are to be motivated by friends in the community - and vice versa. This seems reasonable as more work means more interaction with other people, increasing the likelihood of establishing friendship: "There are pretty strong social ties. For people that are there every day [on IRC channels and mailing lists], you know what's up with them. I mean, you haven't always met them in person, but you know their online presence, and you know them and what they are doing." (interview #9 – openSUSE manager, 11.10.07).

Whatever is the correct interpretation above, it is very interesting that community members explicitly value the relationship to the *product* higher than the social relationships surrounding it. There is clearly a highly motivational aspect related to the identity of the technical product itself which surpasses other explicit reasons for participating. I see this as one of the most important findings of this thesis, and will look closer at the role of the openSUSE object as a unifying identity-carrier, as we move into the next chapter.





## Chapter 5

# Analysis and discussion

---

We have seen how Novell/SUSE has opened up their development process to facilitate a developer-community external to the company. In some respects they are extending their own organization, but at the same time they are keeping externals at arms-length by not granting them authority to commit code to the system. Although these external developers are not part of the Novell organization, they are nevertheless united in a collaborative network that is linked to this organization.

There are several questions that arise concerning the topic of firm-sponsored communities, which I will make an effort to answer in this chapter. The first set of questions are related to my first research topic: *What is the (theoretical) distinction between the sponsor firm and the sponsored community?* To answer this question, we must try to explain the organizational phenomenon we are witnessing in this case. Can we understand Novell and the openSUSE community as one and the same organization, or as two separate ones? Is it even possible to regard the openSUSE community as an independent entity? In the following I will argue they should be regarded as separate autopoietic systems with different properties. The second issue that arises is the question of how Novell and the openSUSE community are connected? Since there are several issues that may cause difficulties in the collaboration, I will discuss which factors it is that makes this co-existence possible. This is more precisely formulated in my second research question: *What is the nature of the relationship between the organizational and the external interactive system, and how are the two systems bound together?* The third set of questions concern the future development of the openSUSE project. How will the community evolve in relation to Novell? What are the possible scenarios? As this type of community only has existed in a limited amount of time, little empirical

experiences exist to provide answers. The research on the area is also limited (O'Mahony, 2007a; J. West & O'Mahony, 2005). My hope is therefore that the theoretical framework used to address the previous questions may shed some light on alternative future scenarios. While addressing my two first research questions, I will simultaneously be pursuing my last research question as well: *How may the case of Novell strengthen our understanding of the theory of autopoietic systems and the theory of boundary objects, separately and in combination?* In the next chapter I will try to sum up some of the most important contributions this case has provided to these theories. First, we will start off by making sense of Novell and the openSUSE project.

## **Theorizing Novell and the openSUSE project**

Novell initiated the openSUSE project in august 2005, and at the time of writing it has existed between 2 and 3 years. In regards to the original road map that was laid out for the project, the launch happened prematurely resulting in a community project that contained the bare minimum of elements necessary for an open source community. The community was bootstrapped upon the existing beta-testers already affiliated with SUSE Linux, and the media-attention following the announcement of the project drew more interested users to the site. The openSUSE community has evolved and grown substantially since its birth, primarily in numbers of users and developers affiliated with the project. The openSUSE Build Service represents the largest technical and organizational innovation within the community, and has shown a strong and interesting development. The openSUSE project has matured to the point where it now includes reliable development tools, established communication channels, a sustainable amount of external developers and contributors, and a large user-base setting expectations for the product. Some representative governance structures such as the openSUSE board have recently been established, and the GPL licensing scheme (that has been present from the start) creates a platform of trust upon which the voluntary community of contributors may continue to grow. In terms of product development, the community-building has started to yield some results, especially in the area of testing, bug-reporting and translation. A fairly little amount of code is contributed by the external community, compared to internal contributions, although the last year has provided examples of product-innovation born and bread in the external community. Novell have however made an effort to create a transparent development process, where most communication is public and contributions are appreciated.

### **Defining systems**

Novell-engineers and openSUSE managers identify themselves as community members, and are acknowledged as such by external contributors. Managers of the openSUSE community are making an effort of erasing the distinction between the company and the community, as this manager illustrates: “When we have internal meetings and we say “the community”, we try to

avoid that or we try not to separate us from the community. Because we are also part of the community.” (interview #10). What are really the boundaries between Novell and the external community? Is it possible to claim that they are erased? Drawing upon autopoietic organization theory, I would argue that the latter is unlikely with the current situation.

In Luhmann’s terms, Novell represents an organizational system. In his view such a system consists of a network of *decisions* – a specific form of communication. All social systems are autopoietic, meaning that all communications are enabled by previous communications, and again make future communications possible. This idea of recursivity creates a link between process and structure (Bakken & Hernes, 2003), and explains how an organization shapes and reshapes itself over time. Novell has undergone a large transition, and decisions have created several changes in the organization. The decision of pursuing a Linux operating system led to the purchase of the SUSE Linux and the Ximian company, which again was a precondition for establishing the openSUSE community. Decisions within the system also involve communications at a more detailed level, such as decisions concerning the next feature to be implemented in a component of the software product, or even the daily commit of a piece of code into the code base through Autobuild. An illustration of Novell as a decision-network is provided in figure 11 below. Since it is an organizational system, it is represented as a square in the figure. Membership in the system is linked to positions that take part in this organizational decision-network. So where does the openSUSE community fit in relation to this system? When an organization is defined as a decision-network, it becomes clear who is *not* part of the system: those who can not make decisions within it. Luhmann’s theory therefore distinguishes Novell as an organization *apart* from the voluntary openSUSE community, which must thus be defined as something separate that exists in Novell’s external environment.

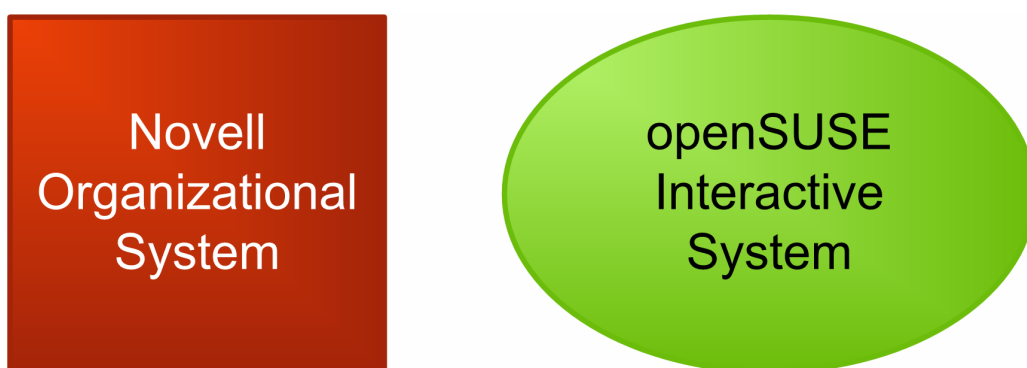


Figure 11. The Novell organizational system and the interactive openSUSE system

Linux product-development in Novell does however happen in collaboration with external members of the openSUSE community, and these *do* have a role in the work on the product. The openSUSE community is a heterogeneous network of people surrounding the development of the

openSUSE distribution, who interact closely with the organizational system. There is a strong internal network of communications within this community, facilitated by the electronic communication channels. As such, this chain of communicative events in the openSUSE community is a social system *in its own right*. The openSUSE network might not have the qualities of an organizational system, primarily since there are no decisions or positions in the system (all decisions concerning the openSUSE project are made by Novell). However, the active openSUSE community may be regarded as an *interactive* system, following its own autopoietic development. For this reason, it is illustrated as an elliptical to differentiate it from the organizational system. Although the people involved with communications in this system are not physically present with each other, Michéle Morner (2003) successfully argues that open source projects may act as interactive systems, since communications flow in public channels across the Internet: “In principle, they have the possibility to follow each and every communication because of its documentation in mailing lists or newsgroups” (2003, p. 264). Due to the fact that communications are accessible to everyone, modularized by topics and are reproducible, she argues further that the stored communication contributes to the *stabilization* of the system. Drawing upon Luhmann, she shows how these features contribute to the system’s *communication connectivity* and *systemic memory* (ibid). This is normally an untypical feature for interactive systems, but explains how a community of voluntary individuals with no formal organization<sup>16</sup> may *persist over time*.

One might want to note that the interactive openSUSE system is not identical with the openSUSE community (whoever that may be). With the framework of autopoietic system theory, we are provided with limits to the parts of the community that are active in the development of the distribution. This helps us solve the issue of identifying the “contributor-community” stated in the previous chapter. Since the system is only based on *communicative events*, the large numbers of users whom otherwise do not communicate with the system are defined as external and part of its environment. Downloading the distribution for personal use does not count as a communication, as only one person is involved. According to Luhmann, even the active contributors – as in the people – are not part of the system. However, we can say that communicative events that constitute the system are *linked to* the senders and receivers to the system (as in the people). Thus, we can continue and say that the interactive openSUSE system is equivalent to the active individuals in the openSUSE community, by means of the communications that are made by these participants. Since people are not part of the system, it is

---

<sup>16</sup> One might argue that Novell, as a sponsor of the community, provides the necessary formal organization to the project. This is however no more true than for any other autonomous open source project, as Novell has no “jurisdiction” outside its own boundaries to, for example, assign people to tasks. All the organizing within the openSUSE community is done in a voluntary manner; community members must accept Novell’s suggestions in the same way as any other independent individual.

possible to say that “the community” may hold any desirable definition (i.e. “anyone who is a user of openSUSE and has an interest in its development”), irrespective of the interactive system involved with product development.

### **System connectivity**

The organizational and the interactive system are both active in the development of the openSUSE distribution, and thereby the enterprise Linux distribution as well (although the interactive system relates to this more indirectly). These two systems are balanced in collaboration on the openSUSE product, somehow. It is clear that these systems are empirically intertwined, but does Luhmann’s theory support such an understanding? I argue it does, as it is not uncommon that various types of social systems overlap (Jönhill, 2003). For example, organizations are crucial for binding different functional systems (i.e. politics and economy) together, as they may inhabit several societal systems simultaneously. Furthermore, Luhmann’s insistence on leaving the individuals outside the systems, allow them to move about and take part in communications within a several systems. It is not as if participation within one system keeps you out of others. The theory can then explain how an employee in Novell can take part in the openSUSE interactive system and other open source projects, while still making communications within the Novell organizational system.

Autopoietic systems theory does not exclude the possibility of participation within several systems, or tight collaborative relations between them. To the best of my knowledge, it may however seem that the theory suffers from a lack of explaining *how* systems are linked together. Connectivity between autopoietic systems is typically referred to as “structural coupling” (Brans & Rossbach, 1997), but Luhmann’s literature seems to exclusively focus on connectivity between societal subsystems:

“Structural coupling is a relationship between systems with each of them belonging to the other’s environment. It involves a system making its own complexity available for the constitution of another system and vice versa. The systems involved in the relationship can take the existence of the other systems for granted and thus concentrate on their own tasks. All the major functional subsystems of society – politics, law, economy, and science – are structurally coupled in this way” (Brans & Rossbach, 1997, p. 426).

Another dimension of structural coupling Luhmann has investigated is the connection between systems in different domains, such as the physiological, psychological and social systems. (Mingers, 2003). However, I have failed to locate descriptions of structural coupling between social systems themselves, at a micro and meso-level. At this level, the perspective is usually that of *one* autopoietic system in relation to its environment. The Novell and openSUSE system are closely intertwined, but how can we explain this connectivity? In the following, I will argue that other theoretical elements such as boundary objects could be a useful supplement Luhmann’s

theory, by offering more explanation to how systems can be bonded together. Pulling in the theory of boundary objects also adds more normative explanatory power to the analysis, which we are deprived of with pure autopoietic theory. I would argue that discussing collaboration between actors in the social world of open source - or any other context for that matter - with disregard to the interests that the various stakeholders represent is problematic, as this would ignore a very important aspect of the organizational reality. Luhmann's theory does not deny the existence of rationality within an organization, but can not explain what this rationality consists of and thus which direction the organization is headed:

“We do not preclude that organizations avail themselves of certain ends (“Ziele”), and rules in the acquisition of means to achieve ends (e.g. views concerning costs), in order to identify themselves. But we leave this question to the clarification of singular cases (...) and maintain only as a basis that an organization exists only as long as its autopoiesis continues and decisions are reproduced from decisions” (Luhmann, 1988, p. 34).

Before going deeper into this analysis, I will first have a closer look at the nature of the boundaries between the two systems and what enables them to interact with each other.

### **The organizational membrane**

It may be too simple to say that the organizational and the interactive system are merely overlapping. If we compare Novell and the SUSE company from five years back and today, something has happened with the organization itself, or rather, the boundaries of the organization. Earlier, the development process was closed off and there was a clear distinction between the world inside SUSE and the world on the outside. Access to the source code was only given after the software was fully developed and released. This is similar to the old Novell organization as well. The organization closed itself off as many proprietary companies do. John Mingers illustrates how organizations erect these barriers, in general terms (2003, p. 115):

“The organization is in a continual process of re-creation of itself anew as communications generate further communications and thereby internal structure. It is the organization that defines its boundaries through its communications and thereby closes itself off, generating a quite **impermeable barrier** for the environment” [emphasis added].

With the openSUSE project, Novell started to offer transparency into the development process, even allowing external influence on the product. Today internal developers communicate extensively with external contributors, and may even receive code contributions from the outside. The boundaries of the organization have changed, and are no longer *impermeable*. What is the nature of the current boundaries?

*Permeability* is an expression borrowed from chemistry, and is used to describe the ability of a substance to let a liquid or gas pass through. In biology, for example, it is common to discuss the

permeability of cells' plasma-membranes. Within organization theory there are few words that describe the composites of boundaries, and it may therefore be fruitful to borrow some vocabulary from this metaphor. The cell's plasma-membrane is described as *semi-permeable*, as proteins located in the cell's membrane allow the transportation of certain ions such as Sodium and Calcium in and out of the cell, while others substances may be rejected. The plasma-membrane controls the traffic selectively, and is therefore also said to exhibit *selective permeability*. A special form of protein, named aquaporins<sup>17</sup>, enable the continuous and free flow of water molecules through the membrane regardless of this selectivity.

Without drawing this analogy too far, we could say that the organizational boundaries of Novell and SUSE have evolved from being *impermeable*, when product development was done in a largely proprietary manner, to becoming *semi-permeable*, as development has become more transparent and communication concerning product development continuously flows in and out of the company. Influences on product development are also admitted on occasion through these boundaries, but only on a selective basis – Novell has sovereignty over all decisions in the project. Novell are therefore also exhibiting *selective permeability* towards its environment. The communications that flow through Novell's boundaries and the activity that is centered upon them are facilitated by some important elements, namely the boundary objects and communication channels that are located at the intersection of the Novell organizational system and the interactive openSUSE system. We will now move on to see how these operate.

## **Binding systems**

By creating the openSUSE project, Novell initiated the emergence of an interactive system that now surrounds the development of the openSUSE distribution, but which is held at arms-length by a semi-permeable boundary from the organization controlling it. How do these two systems interact? What binds them together? I argue there are three elements that ensure a tight coupling between the two systems. First, several developments tools and models serve as important *boundary objects* between the two systems, enabling joint development on a common product. Secondly, *shared communication channels* are vital in creating transparency and providing access through the boundaries of the systems. Thirdly, the *marginal people* whom have roles in both systems are crucial for balancing the needs of both of them. In the following, I will address these three aspects in closer detail, but my analytic emphasis will remain on the first and third of the three. This does not mean that communication channels are not equally important in bridging the systems, but they do not demand the same amount of explanation. The following discussion will show how Luhmann's theory may be supplemented and strengthened with the theory of

---

<sup>17</sup> The discovery of aquaporins happened recently, and in 2003 Peter Agre received the Nobel prize in chemistry for its discovery.

boundary objects. Moreover, it will show how the theory of boundary objects not only connects individuals and groups, but also *social systems*.

### **Boundary objects**

I have previously discussed the importance of boundary objects in collaborative work, based on two crucial properties: their ability to contain and create coherence among separate interests, and their ability to carry common knowledge and identity at the boundaries between social groups in collaboration. There are several such objects that play an important role in the development of the openSUSE distribution. We will immediately start off by looking at one such example, before returning to a discussion of their role in the collaboration.

#### **A boundary object example**

In all software development work, bug-tracking tools are crucial for organizing quality assurance processes. Bugzilla is the name of the tool used in openSUSE, and is itself an open source software tool that is widely used and familiar to most open source developers. Bugzilla keeps track of bug-reports, which normally represent notifications of an error in the software. Bug-reports are created retrospectively, typically after a user has tested a software product and detected an error somewhere. The user can then create a bug-report describing the problem, whereby a developer is assigned to finding a solution. A discussion may then go on in the bug-report between the user, the developer and other users that have an interest or opinion in the matter. Discussion entries may include attachments, such as screen shots of the problem or a patch of code that might fix the problem. A generic model of a bug-report is provided in figure 12.



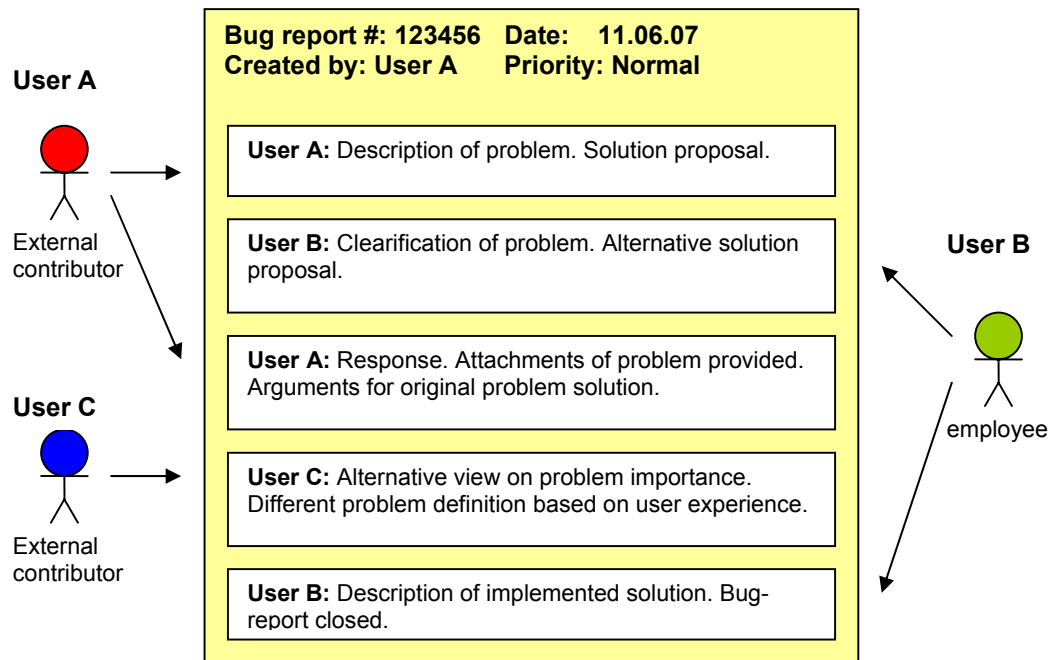


Figure 12. Model of a bug-report.

Although most bug-reports are rather non-controversial and straight-forward descriptions of a problem and its possible solutions, there may be different interests present and discussions can arise. Let us imagine that the following interests were present in the model above:

**User A:** Primary interest: get annoying problem solved. Secondary interest: provide a constructive contribution, and gain recognition for having identified something important.

**User B:** Employee. Primary interest to solve problem as quick as possible with least amount of time and cost.

**User C:** Interested in exposing himself as a competent user/developer. Secondary interest to solve problem.

The example is strictly hypothetical<sup>18</sup>, but shows how the boundary object may contain multiple interests. Everyone wants the problem solved, but have other goals as well. The object is thus able to support simultaneous translations (Callon, 1986), as it does not fall apart even if there should be conflict (translations are attempts at creating an alliance by rallying the others to your problem definition). Defining the problem and suggesting a solution is a process of creating an obligatory passage point; the necessary path to take to achieve the common goal. In the case above, there might be three possible passage points; one for each user. Who will succeed in rallying the others

<sup>18</sup> I could have used a real example from Bugzilla to illustrate what a bug-report looks like and how it contains communication, but it would not be able to reveal the interests behind each statement. For that, a deeper analysis and empirical study would be needed. Therefore, a hypothetical example will have to suffice to illustrate the point.

will depend on their arguments, strategies and positions. The original problem owner, user A, may use screenshots as interessement devices to enrol others into understanding how this problem must be solved. User B, on the other hand, is the user with the authority to actually commit a code solution to the code base, and thus holds the upper hand. If however user A or C provides a probable solution that saves him work-effort, he will be likely to become enrolled and accept it.

While the discussion concerning the problem and its solution takes place *within* the bug-report, talk about the problem may still happen *outside* the object itself. Once placed in such a report, a problem in the software becomes an object that can be categorized, referred to, assigned to people, create discussion and trigger decisions. Below are two extracts from public openSUSE project meetings (organized every second week). Both employees and contributors participate, and one of the objectives in these meetings is to follow up work on bug-reports that concern non-technical issues. The first extract is an example of this process.

```
18:09 @<moderator>      Bug #343444
18:10 @<moderator>      contributor1: what about that one? :)
18:10 <contributor1>     never heard about that
18:10 @<moderator>     well read your emails dude its assigned to you
18:11 @<moderator>      caught
18:11 <contributor1>     yes.
18:11 <contributor1>     hmm
18:11 <contributor1>     ok, Asche on my haupt
18:11 <contributor1>     sorry, not yet on my radar
18:11 @<moderator>      okay
18:11 @<moderator>      no change then
```

*Extract #1 from IRC project meeting.*

This extract starts with reference to a specific bug-report that (#343444), and the moderator asks the contributor that is assigned to solve the problem about its status. Having forgotten all about it, the contributor admits he has not been able to do anything about it yet, and is “caught”. He has, unfortunately, no change to report. This trivial extract illustrates how bug-reports are used to define and frame a problem that may be quite complex into an object that is easy for everyone to relate to. They contain shared knowledge and have a common identity, and are equally instrumental for contributors and employees alike in organizing their work. This second extract also includes a short discussion concerning the priority of a specific bug-report (my emphasis):

```
13:18 <@moderator>      Bug #293726
13:18 <bugbot>          openSUSE bug 293726 in openSUSE.org "Creation of Babel
wiki"
13:19 <@moderator>      that one also desperately needs closing
13:19 <employee2>       wiki update is more important for me
13:19 <employee2>       after that we can look into this
13:20 <@moderator>      c'mon. just be serious. its rotting since 6 months now
13:20 <@moderator>      there will always be something more important
13:20 <employee2>       it would be a nice to have, but we are not dieing
without it
```

|                    |                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 13:21 <employee2>  | and since john mentioned some problems with the setup, i will not do a thing before the update ... new software new problems ... hopefully |
| 13:21 <@moderator> | okay                                                                                                                                       |

*Extract #2 from IRC project meeting.*

When the moderator says that the bug-report “needs closing”, it means that the problem somehow needs to get solved so that the bug-report no longer is active. He also argues that it has been active for 6 months and needs to be solved soon. The employee however finds other tasks more important, and successfully argues that the problem should be postponed until a software upgrade has been done. This discussion only concerns the priority and time-frame for solving the problem. What the problem *is* and how it should be *handled* is captured within the bug-report itself, as shown previously. This extract shows how a software problem becomes a task. All the active bug-reports put together is therefore also understood a grand *todo-list*, perhaps more so by employees than external contributors. Since the employees are the ones officially responsible for dealing with the bug-reports on “their area” of work, the bug-reports can have another meaning for them (in addition to serving as collaborative objects and stages for conflicts of interest): Several employees I spoke with mentioned that one of the only ways of receiving feedback on their work was through Bugzilla. If there were few bug-reports on your project after a software release, you have done a good job:

[Myself:] “In what ways do you receive feedback on what you do?”

[Packager:] “Ha! Usually, never! But there is an indirect way: Bugzilla. If you have a lot of bugzilla’s after release was done, then you know your work wasn’t good. But usually, I hope I do my best and I don’t get really much feedback” (interview #16).

Bugzilla is a collection of bug-reports, and includes many thousand entries. As a boundary object, Bugzilla can be described as a *repository* of objects (Star & Griesmer, 1989), and represents one of the important characteristics of the openSUSE project which distinguishes the present open development from the previous closed process in SUSE. When Bugzilla was “opened up”, Novell managed to create a powerful boundary object that has proven crucial for binding the interactive openSUSE system with the organizational decision-network of Novell.

### Other objects

In addition to Bugzilla, there are other examples of boundary objects found in the collaboration between Novell and the openSUSE community, including the openSUSE Build Service (OBS) and the Factory code base. The OBS is a repository of software projects, where each project may involve a number of developers ranging from one to several hundred. For each software project, the OBS includes supportive infrastructure for project management, communication and discussion. As a whole, the OBS is a remarkable boundary object. It holds room for unlimited

interests and translations. If a dispute breaks out in a software project, one of the parties can just fork the code, start their own software project in the OBS and rally their own developers. The OBS is a tool for software development, a showroom for projects and an infrastructure for alliance-building.

While the example of Bugzilla and the OBS so far has shown us how boundary objects manage to keep people together in collaboration - despite any boundaries or conflicts of interest - the Factory code base, on the other hand, displays another quality of boundary objects; namely that boundary objects are able to *connect and separate* people, groups and systems *simultaneously*. The Factory code base represents the absolute latest development of the distribution, and is therefore an object that all developers relate to. However, Factory only contains the official or accepted development projects; it does not contain all the various private development initiatives found in the OBS. There is a clear distinction between who may alter or impact the content of the Factory code base. Only employees have direct access to commit code updates that are eventually shuttled into Factory. The Factory object therefore illuminates the boundaries between the Novell organizational system and the interactive openSUSE system, while simultaneously binding them together.

As a contrast to the mentioned boundary objects, we can look at other collaborative objects which can *not* be considered boundary objects. For example, ‘Autobuild’ as a model of development and technical artifact is without doubt an important collaborative object, but this object remains within the Novell organizational system. It is surely an important boundary object between groups *within* Novell, but at this level of analysis it must be regarded as an internal naturalized object (different departments and teams within Novell is not our focus). As a naturalized object, having knowledge and familiarity with the object is an indicator of membership in the group (Bowker & Star, 1999). This is quite true in Novell’s case: You will not get very far with your work in Novell’s Linux R&D without any idea of what “Autobuild” means, nor does anyone outside Novell’s Linux R&D have any clue of its meaning (there is in fact some confusion about this question internally as well). Returning our focus to the relationship between the Novell organizational system and the openSUSE interactive system, we may say that the boundary objects, put together, provide a framework that holds all the actors together - as individuals, groups and systems, while simultaneously supporting the qualities that separate the same people.

In sum, we have identified a few objects situated at the center of the Novell and openSUSE collaboration, and illustrated how they contain qualities as boundary objects. At this point I would like to start a theoretical discussion on the *role* of boundary objects, spurred by an observation made in the case of Novell and the openSUSE project: Neither the mentioned objects nor the theory of boundary objects itself can explain *why* the actors come together to

collaborate in the first place. What is the normative foundation for their joint venture? Star's theory would have us look other places for answers to this question, perhaps diverting us to economic business models and the open source literature's egoistic, altruistic, intrinsic and extrinsic incentives. I however believe that an answer to this question may exist within the boundary objects themselves. To illustrate: the most powerful boundary object of all in the collaboration in Novell is not one of the previously mentioned objects, but the end-goal of the collaboration itself: *the openSUSE distribution*. This is what all the collaboration is about; it is where all stakeholders hold their primary interests. It is also what drives all the actors together to collaborate. Before moving on with this discussion, it could be wise to sum up the simultaneous claims I am about to make: First, I am saying that the openSUSE distribution holds a quality as a boundary object that is different from other boundary objects such as Bugzilla, the OBS and Factory; Secondly, this quality is the openSUSE distribution's ability to create a normative motivation for collaboration among the actors; Finally, the two previous claims imply that a discussion and extension of Star's theory of boundary objects is desperately needed. Unfortunately, it is hard to explain everything at once, so I will have to ask the reader to be patient and bear with me while we walk through this line of thought. There are several questions that need to be answered (preferably at the same time), which I will address in the following order: First, how can the openSUSE distribution be a boundary object in the first place? Is it not just the product of collaboration? The second question I will pursue is this: What would the motivating aspect found in the openSUSE distribution mean for our understanding of boundary objects in general? Thereafter I will show how this claim is supported by some empirical findings, before I lastly look at what theoretical foundation we may find that may explain what I have witnessed in the case of Novell and the openSUSE project.

### **The openSUSE object**

The openSUSE distribution is the product and the end result of the Linux development in Novell. I also find that it is an important boundary object situated at the intersection of all groups involved. It is reasonable to question whether the product actually can be a boundary object. However, I do not claim it is the distribution *as such* that should be considered the boundary object, as in the DVD with the running software on it. Rather, it is the *model* of what the distribution is and should be, as it is perceived among all collaborative participants. It is the idea of the product, and the meaning it holds to everyone involved. It is the word that is used whenever developers talk about "working on openSUSE", or the meaning they put into this specific software project when they compare it to other operating systems and software projects. In Star and Griesmer's terms, the openSUSE object represents an *ideal type* of boundary object: "It is abstracted from all domains, and can therefore be general enough to be applied in many contexts" (ibid, p.410).

Showing how the openSUSE model has characteristics of a boundary object is not very difficult. It is definitely a shared object between all stakeholders in the collaboration, and has a *mutual identity* among them all. Whether you are talking to a veteran Novell-employee from the “old” company, one of the executives at the headquarters, an openSUSE community developer or an un-experienced openSUSE user, they will most likely share a general understanding of what openSUSE is, as exemplified in figure 13 below. The figure contains a few basic keywords I have encountered in conversations with developers concerning the identity of the openSUSE distribution (but it is by no means any official description):

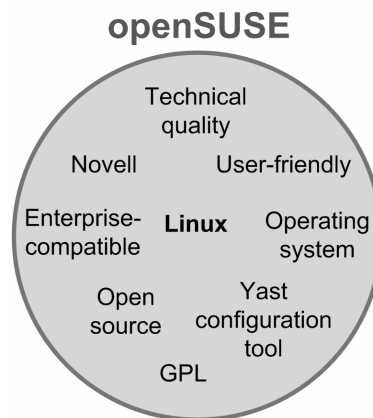


Figure 13. Keywords in the openSUSE object

Moving on, the object also contains *common knowledge* available to developers in all domains. This knowledge is directly linked to the physical distribution, as it is embedded into the software code. However, a developer does not have to look into the code base to know what the main packages in the openSUSE software architecture are, or which features and qualities it includes. The openSUSE object also holds knowledge about its historical development, which is important in forming its identity.

While the openSUSE object is able to hold many common qualities, it is even more adept at handling diversity and simultaneous translations. Consider, for example, the following statements. The purpose of the openSUSE distribution is to be:

- “a testbed for commercial products”
- “the best open distribution”
- “a development model for Novell”

These are just a few statements I have encountered in my conversations and observations with various informants. It would be possible to list a large number of interests related to the openSUSE distribution. The intriguing feature of the openSUSE object is its ability to handle many different interests such as these at the same time – they do not exclude each other! For

example, the object can support the interests of using the openSUSE distribution to appropriate returns in the software market, support proprietary enterprise solutions running on top of it *and* at the same time advocate principles of free software distribution. The object does not necessarily *harmonize* the interests; proponents of various views may be in hard conflict with each other. However, the object is able to make these interests *compatible* with one another. Note that there is a difference between which interest are present in the openSUSE object and those which may be *physically visible*, since the aspects and interests that are enacted *in practice* will depend on who comes out best with the multiple and ongoing translations surrounding the object. For example, we may recall that Novell’s first release of the Linux product was labeled “Novell Linux Desktop”. After this release, former SUSE employees in Novell and existing SUSE users where successful in establishing the “SUSE” brand as an obligatory passage point for the future development of the Linux product, which has lived on in the openSUSE model. Figure 14 below is an illustration of how two interests are present in the openSUSE object, drawing respectively upon the normative structure of the commercial and the open source software model<sup>19</sup>:

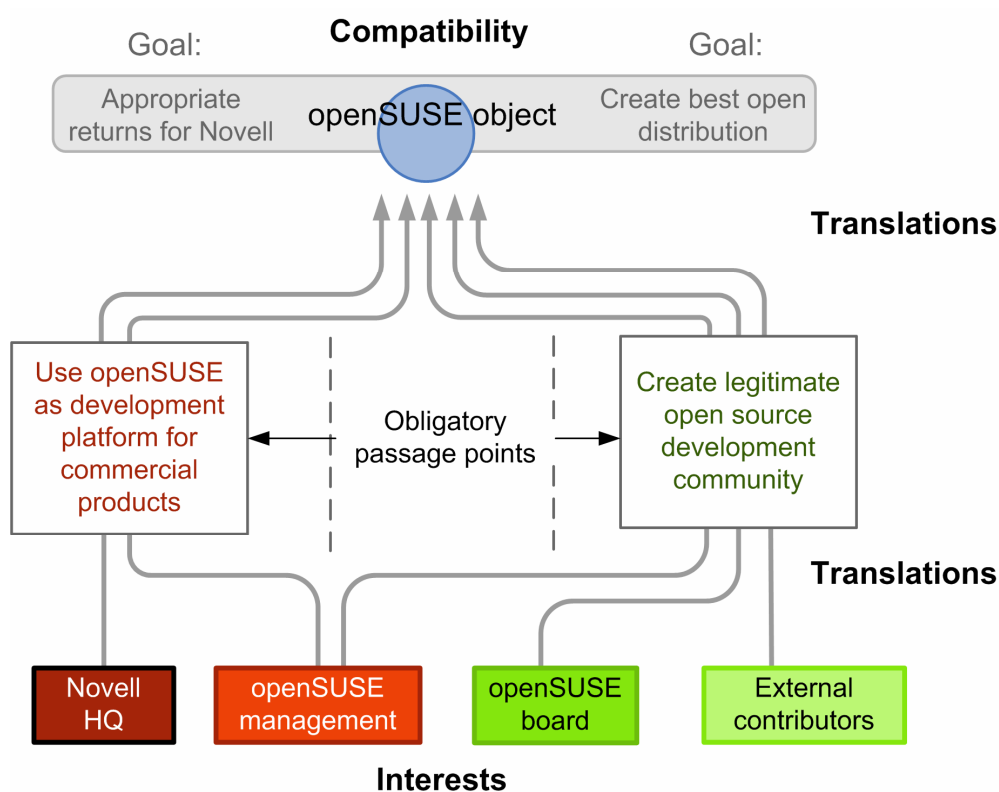


Figure 14. Illustration of translations in the openSUSE object. Model derived from Star and Griesmer (1989).

<sup>19</sup> Note that the model above is merely an illustration of how the openSUSE object may support n-way translations, and should not to be taken too literally. Other interests than the two illustrated surely also exist. The alliances that are drawn are also only used as examples, and may not represent the full picture.

As we have seen empirically, the obligatory passage points that are established by advocates of the interests present in the model above are able to exist simultaneously. For many developers that “choose” the path on the right, the general sentiment is that the left side is a necessary path to take *as well*, and does not interfere with the work of creating a strong open distribution. This open source advocate (employee) says he does not mind that Novell sell expensive proprietary products on top of the openSUSE product to enterprise customers:

“No, these are not free [Groupwise, ZenWorks and other proprietary Novell products]. And actually I am not sure if these need to be free. Because most users for the desktop system don’t need it. **And we have to make money somehow. And when the enterprise customer pays for it it’s OK for me**” (interview #10 – my emphasis).

### The role of the object

Now that we have shown how the openSUSE object is a legitimate (and important) boundary object, we can return to the unique features of this particular object and the point I am trying to make. I have mentioned that the openSUSE object has an ability to draw the collaborative actors together, and I further claim that it creates a fundamental motivation for the entire project. To illustrate this point I would like to revisit the data from my contributor-survey, and specifically figure 10 in chapter 4 (p.88). The figure shows that the most motivating factor among community developers is that they all believe that openSUSE is the best Linux distribution; this is what they claim is most important for their willingness to participate in the project. Whether openSUSE in fact *is* the best distribution or not, is beside the point. The impression that the openSUSE distribution holds undisputed quality is most likely created by the developers’ own experience with the software, along with other community members’ opinions that confirm this impression. This subjective experience then feeds right back into the identity they associate with the openSUSE object, which in turn motivates them to continue pursuing its development. This finding shows the importance the openSUSE object plays for the motivation of the contributors in the community.

Now is a good time to raise this discussion to a more theoretical level. What are the characteristics of the openSUSE object that makes it different from the other objects we have discussed? And what does this mean for our understanding of boundary objects? I believe we are witnessing that boundary objects can play different *roles* in collaboration. Bugzilla serves a different purpose than the openSUSE object in the cooperation between Novell and the external openSUSE community. I believe this finding addresses a weakness in the theory of boundary objects, as there is a lack of distinction of the object’s individual role in the collaboration effort. It is clear that there is a difference between objects that represent the end-goal of the collaboration effort, and those that merely serve as a tool for reaching other aims among the respective collaborative



groups. I would therefore propose a distinction between those objects that are *means* and those that are *ends* of the collaboration effort, and suggest to name the former *supportive-objects* and the latter *target-objects* (since their development is the target of the collaboration). Bugzilla and the OBS are examples of supportive boundary objects, while the openSUSE object is clearly a target-object. This understanding provides a new impetus for explaining the motivating qualities of the openSUSE object: Since this object represents the ultimate end-goal of the collaboration, this role is obviously important in explaining one of the driving factors behind the project. We may understand the importance of this quality by reviewing other some theories that address some similar aspects; the concept of *epistemic objects* (Knorr Cetina, 1997, 2001; Rheinberger, 1997) together with our knowledge of *anthropological symbols* may provide strength to our understanding of target-objects.

### **Target-objects: towards a deeper theoretical understanding**

Target-objects hold a property that the supportive-objects do not; they represent a *self-reinforcing motivational factor* in collaboration. To emphasize this point, we may borrow inspiration from another theoretical strand on objects in collaboration: The notion of epistemic objects in scientific work (Rheinberger, 1997) – also called knowledge objects (Knorr Cetina, 1997) – emphasizes the role of the object as the goal of collaboration. Since the object is the end-result, it has not yet become what it later will be. Epistemic objects are therefore “as much defined by what they are not (but will, at some point, have become) than by what they are” (Knorr Cetina, 1997, p. 15), and they “embody what one does not yet know” (Nicolini et al., 2007). This creates a “lack of completeness” in the object that ultimately spurs a desire among the people working on it. Knorr Cetina describes this as the object’s *structure of wanting* (1997). The desire of working on the development of the object is common for all the actors, and is therefore one of the means that hold them all together. The openSUSE object contains these qualities. Consider, for example, this statement from a community developer (written in an open text field under the question “Why do you participate in the openSUSE community?” in the contributor survey): “[Because] I *want* it to be the best distro... even though I think it still falls short in some areas.” This statement includes *his* emphasis and illustrates his desire to make openSUSE the best possible Linux distribution. Although he might not believe openSUSE actually *is* the best distribution at the moment, it still drives him to contribute with what he can to reach that goal. It is a perfect illustration of how the openSUSE object contains the structure of wanting characteristic to epistemic objects. I therefore conclude that this is an aspect that can be generalized to all target-objects, as they all represent ends in collaboration. According to Nicolini et. al, any object that is “in the process of being materially defined, functions as an epistemic object”. This could seem problematic for the case of the openSUSE object, since this already *is* materially defined. Knorr Cetina claims however that objects which are “simultaneously things-

to-be-used and things-in-a-process-of-transformation” (1997, p. 10) must be included in the category of epistemic things.

A second strand of thought that gives credit to the motivating capabilities of target-objects can be found in Emile Durkheim’s classical work on social ritual theory (1965 [1912]). His theory emphasizes the role of objects (also referred to as symbols or totems) as centerpieces of attention among social groups:

“Placed thus in the center of the scene, it becomes representative. The sentiments expressed everywhere fix themselves upon it, for it is the only concrete object upon which they can fix themselves” (Durkheim, 1965 [1912], p. 250).

When people come together in social rituals<sup>20</sup>, the interaction generates sentiments – a “sort of electricity” (Durkheim) or what Randall Collins calls *emotional energy* (Collins, 2004, p. 38). We then use symbols and objects as representations of the abstract sense of collective consciousness that is created when we as collaborators come together, “[f]or we are unable to consider an abstract entity the source of the strong sentiments which we feel. We cannot explain them to ourselves except by connecting them to some concrete object of whose reality we are vividly aware.” (Durkheim, 1965 [1912], pp. 250-252) The objects are thus charged with the sentiments that come out of our meetings:

“What is mutually focused upon becomes a symbol of the group. In actuality, the group is focusing on its own feeling of intersubjectivity, its own shared emotion; but it has no way of representing this fleeting feeling, except by representing it as embodied in an object” (Collins, 2004, p. 37).

The powerful capability of Durkheim’s symbols are that they are able to unleash the sentiments they are charged with when the group is dissolved and individuals are apart from each other: “it is as though the cause which excited them in the first place continued to act.” (Durkheim, 1965 [1912], p. 265). In this way the symbol has the ability to continue to unite the group when they are *not* together. Individuals are drawn towards the objects because of what they represent and the sentiments they contain. Lars Risan (Forthcoming) has studied how this social mechanism is manifested in the Debian Linux community, where he finds a “love for Debian” among the Debian developers. He describes this emotion as an “expression of sentiment, desire and belonging that goes far beyond economic rationality” (Risan, Forthcoming, p. 26). We can relate this back to the statement from the openSUSE contributor above, who desperately *wants* the openSUSE distribution to be the best distribution. It is likely that his emotional attachment to the openSUSE object is what explains this desire, rather than any rational motivation.

---

<sup>20</sup> Note that a social ritual does not need to have any religious association. Rituals are tied to human interaction and have a recurrent nature. They can exist on a large or small scale, from the gathering of a crowd (e.g. a demonstration), a board meeting or even a handshake (Collins, 2004, p. 34).

Can we append the capabilities of Durkheim’s symbols to our understanding of target-objects? The theory certainly fills a critical gap in our thinking so far: namely accounting for a larger whole that is capable of holding together two geographically distributed autopoietic systems – an explanation that gives us *more* than an aggregate of various individual incentives and motivations, and that fits with what we have witnessed in the case of the openSUSE object. In the chapter 2 I mentioned that Star’s theory of boundary objects may need an extension that can explain a larger unity that can hold the collaborating groups together. I argue that the symbolic value in objects such as the openSUSE object is the *manifestation of the symbolic unity within an organic solidarity*, and provides the “glue” to keep it all together despite diverging interests and other conflicts that may be.

It is however important to stress that target-objects and anthropological symbols are *not* one and the same. Durkheim’s symbols are representations; they are placeholders for the sentiments of the collective’s physical presence when it is not there. Target-objects are something *more* than that. The openSUSE object is not only a container of the communities sentiments. Drawing upon our understanding of epistemic objects above, we see that the object *itself* holds an intrinsic motivating quality, in addition to being a carrier of individual and collective sentiments in the sense of Collins and Durkheim<sup>21</sup>.

I would like to make a last note on the topic of target-objects. In voluntary work, individual motivation may be more important than in other communities of work. I believe that what we now know about target-objects addresses a weakness in our research on open source developers and on explaining their motivation to contribute lots of hours of development for no pay. In the motivational grid drawn up in figure 2 in chapter 2 (p.11), I have grouped the various motivations found in the literature within the categories of egoistic, altruistic, intrinsic and extrinsic (the compilation is based on findings in (Dahlander & Magnusson, 2005; Demil & Lecocq, 2006; Hippel & Krogh, 2003; Lerner & Tirole, 2002; O’Mahony, 2003; Raymond, 2001; Stallman & Gay, 2002; Weber, 2004). This grid can account for most answers you will receive from any open source developer (it can for example explain each of the 70+ extra comments on reasons for participation, that were supplied in open text fields by respondents in my contributor-survey). However, I have argued adamantly for target-objects’ importance in creating motivation in collaborative networks, yet this point does not show up elsewhere in the literature referred to above. Why? When individuals are asked about their reasons for doing something, they start reasoning and accounting for the motivations they are able to find within

---

<sup>21</sup> I would like to add that my use of Durkheim’s understanding of symbols is coherent with Knorr Cetina’s concept of object-centered sociality (Knorr Cetina, 1997). Although it is tempting to elaborate on the relationship between them, this thesis does not include enough room for a further discussion of this topic.

themselves, preferably looking for a rational explanation the rest of the world can acknowledge<sup>22</sup>. As researchers, we are also constantly looking for answers *within the individual*. Perhaps also because we are social scientists, we can lose sight of the role of material artifacts surrounding us. I would say that neither of these individual, accountable reasons explain why a developer or user wants to wear a t-shirt with the logo of the software product he is working for, or why he or she gets seriously upset when someone makes a bad remark about the software or say they prefer Microsoft Windows. Individual reasons also fall short of explaining how all these single developers are drawn together and united towards the same goal. As figure 15 above illustrates, the object exists in all grids, and makes the dichotomies between egoistic-altruistic, intrinsic and extrinsic rather irrelevant. The object exists *externally* to all individuals, yet they all identify themselves with it. Hopefully, the presentation and discussion of target-objects has provided some new insight on this area, and shown how an object can provide such motivation.

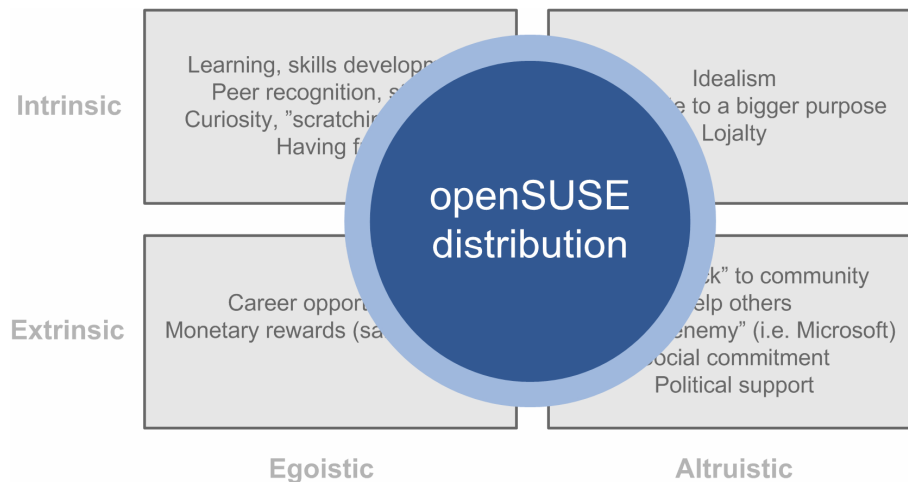


Figure 15. Target-object motivation.

We have now reached the end of this discussion on the role of boundary objects. The case of Novell and the openSUSE project has shown how we can improve our understanding of this theory, with the distinction of supportive- and target-objects. In return, the understanding of target-objects may also provide some advice for Novell: If they want to attract motivated contributors to the community, they need to give as many developers as possible access to the openSUSE object. We will return to this subject in our discussion of future scenarios towards the end of this chapter. Instead, we can sum up our discussion so far: We have seen how the

<sup>22</sup> Having been an active student-spokesman working full-time for over a year at a student-office for no pay, I have first-hand experience with the tedious process of explaining curious friends and concerned family about the reasons for such insanity.

autopoiesis of social systems creates boundaries between the Novell organizational system and the interactive openSUSE system. Boundary objects are situated in-between and ensure system-connectivity, whilst also maintaining separation. Target-objects, on the other hand, contribute to form a unity surrounding both systems, interlocking them together. This understanding above is however not *enough* to explain how the actors surrounding the openSUSE project stay united. Boundary objects are only *objects*; they do not act on their own. In a moment we will therefore look at which *people* play a particularly important role in activating these objects. First, we return to investigate the boundaries between Novell and its outside environment closer, and the channels that make important communication through the organizational membrane possible.

### **Shared communication channels**

Although the boundary objects mentioned are vital for the collaboration, there are other important elements as well. The open communication channels are crucial for providing an arena where the collaboration can take place. The network of electronic communication channels consists primarily of the mailing lists, the wiki, and the IRC channels. These serve important purposes within each system, as mediators of communication. For the openSUSE interactive system they are especially crucial, as they are the only channels of communication in the system and since most members of the community are geographically distributed across the globe without being able to meet physically. Within Novell, other communication channels exist (meetings, coffee breaks, etc), but in Novell's Linux R&D the managers insist on using the open electronic channels for development work in order to create transparency and enable interaction with the external community members. According to my informants, almost all project development is communicated here; exchange of ideas, initiation of new projects, discussion about current and future solutions, problem solving on existing bugs and so on.

On the mailing lists, each participant in the community and the company can configure their own communication channels, related to their own interest. By selecting which lists to subscribe to, they can follow certain threads and topics and ignore others. The IRC channels support synchronous collaboration. The Wiki is especially suited for providing a portal to the other communication channels and for presenting static information, documentation and archiving communications on the mailing lists. Following Luhmann and Michèle Morner, we can say that these shared communication channels contribute to the *stabilization* of the interactive system (Morner, 2003), preventing it from dissolving over time. The key features in achieving stabilization are modularity of communications, accessibility to the channels and information, traceability and documentation of communications. These features are all provided by the mailing lists, IRC channels and the Wiki.

The communication channels in the organizational system and interactive system are shared. Before the openSUSE project was initiated and these channels were opened, communication concerning product development did not cross the organizational boundaries of Novell/SUSE. If we continue the metaphor from the previous section, one might say that these channels of communication are the *aquaporins* at of the organizational membrane. They ensure that communication concerning product development flows freely through the Novell’s boundaries. The aquaporins escape the selective permeability that is exhibited by a cell’s membrane; water can flow in and out whenever it pleases<sup>23</sup>. In a similar fashion, there is no censorship or control of communication on the community channels, anyone can post a statement that is transported throughout the entire network.

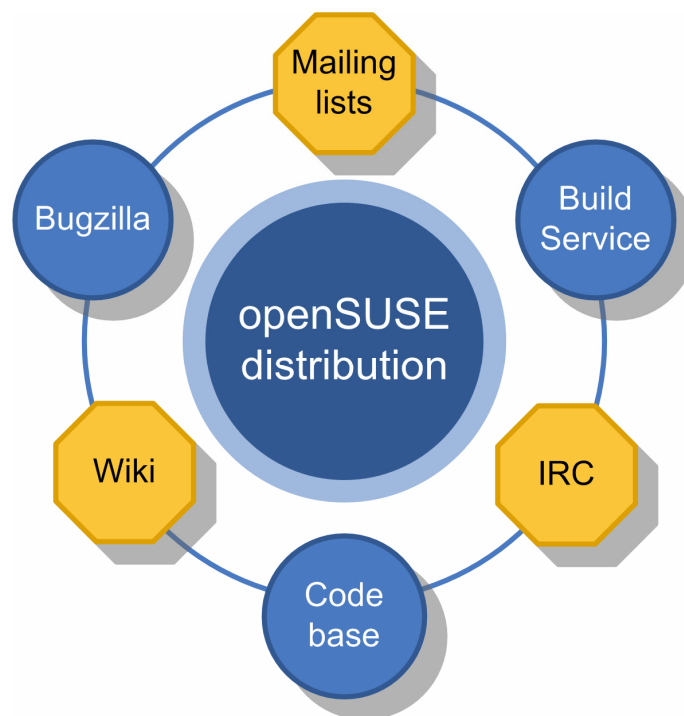


Figure 16. Network of boundary objects and communication channels.

The communication channels and the boundary objects are tied intricately together, as illustrated in figure 16 above. As shown in the example from the IRC chat, communications surrounding the objects often take place on community channels. Another example is provided by the fact that the mailing lists “Factory” and “Build Service” carry the very names of their respective objects. We can see how a set of objects and channels are important in binding the organizational and interactive system together, and form a sort of integrated network. While this set of objects and

<sup>23</sup> Water is the only substance that has is able to flow in and out of a cell unconditionally through the aquaporins, the traffic of other substances are “controlled” selectively by the cell’s membrane.

channels contribute to system connectivity, there is also one last category of forces at work that bind the systems together: the people that communicate within both systems.

### **Marginal people**

The distinction between the interactive openSUSE system and the organizational system in Novell will for some people seem artificial. Or, at least, they would rather argue that the boundaries between them are fuzzy and unclear. These are the people that actively communicate in both systems. They fill positions in the organizational network where they participate in decisions. At the same time they communicate actively with the openSUSE community which they are also a part of. They are on the one side the employees and managers in Novell's openSUSE department that are active in the community structures, and on the other side the external community members that have earned their stars and trust within the organizational system. We might say these are people with *dual membership*. In the terms of Susan Leigh Star, people whom inhabit multiple social worlds are called *marginal people* (Bowker & Star, 1999; Star & Griesmer, 1989), as they live at the margins of several domains. This understanding draws upon Robert Park's marginal man, "the one who has a double vision by virtue of having more than one identity to negotiate" (Bowker & Star, 1999, p. 302). In Novell and the openSUSE project these identities are related to the commercial software model on the one hand and the normative structure of the open source model on the other. Marginal people may typically experience tensions imposed by the multiple membership, and problems of identity and loyalty.

#### **Who are they?**

Within Novell these people are primarily found in Novell's Linux R&D, and may be identified as the most eager internal advocates and participants of the openSUSE community. In total there are approximately 300 people employed within this R&D team, but only a fraction of these may actually fit the description of marginal people. There are many employees that are used to the 'old way' of working in a closed development process, as this manager illustrates:

"I encourage people [employees] to be more outside, I encourage them to communicate upstream and also on the openSUSE mailing lists, to actively participate, but for some of them it takes time. So even if we are doing open source development, a lot of people think in a closed source world, and have trouble communicating with the outside" (interview #14).

From my meetings with individuals at Novell who *are* active on community channels, I found that a common characteristic among them is a background from participating in other open source communities, previous to their employment in Novell. There are also examples of employees whom have never been involved in communities before, but have still become among the most active advocates and participants in the openSUSE community. I argue that the

experience and familiarity with the open source model (as a methodology and normative structure) among the former group has nevertheless been important in order for others to follow.

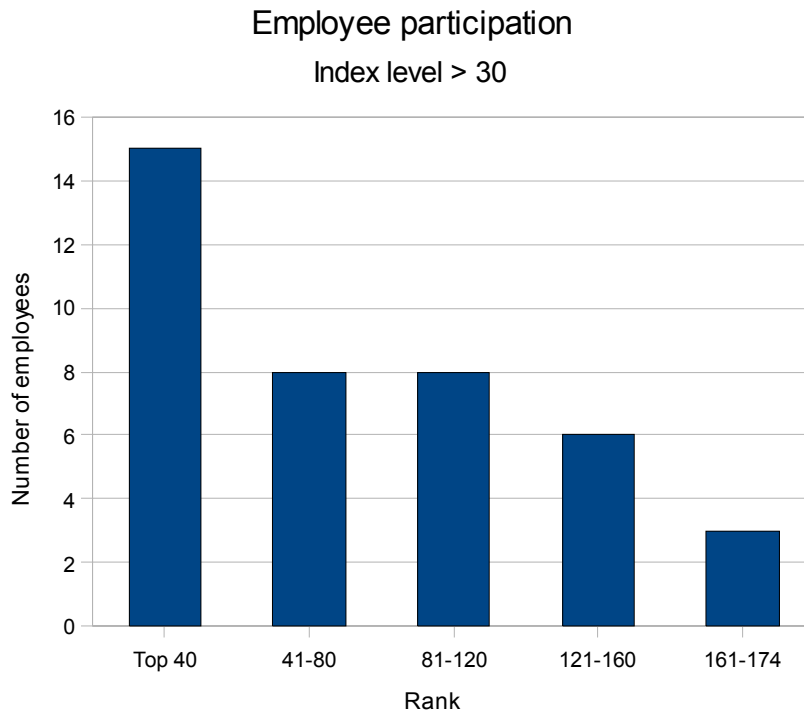
From the statistical analysis of the main developer-mailing-lists we can operationalize an indicator of these marginal people, and locate an approximate number of Novell-employees that may be described as such - in terms of their participation on these communication channels. For each mailing-list participant, the data includes the total number of entries and the number of months the entries are distributed over, by this participant *for each* mailing list (see overview of these lists and their participants in the previous chapter). The data is from the time period of September 2006 to October 2007. Based on these variables, it is possible to create an index in order to rank the participation of mailing list users. The following is an algorithm for such an index, which balances participation over time with amount of contributions:

$$n \sum_{i=0} = x_i^2 + y_i$$

where  $n = 5$  mailing lists (the developer-lists),  $x$  = months participated and  $y$  = number of entries (emails to the list). A participant whom has sent 60 emails to one list over 3 months will gain an index of 69, whilst another participant whom has sent 20 emails over 10 months to one list and 15 emails over 4 months to another will have an index of 151 ( $120 + 31$ ). As the observant reader will notice the number of months you have participated increases exponentially in value over time, meaning that it is more important to contribute over time than to post a large amount of emails within a limited time-frame in order to get a high ranking. I tested the results of this and a few alternative ranking-algorithms with some of my key informants, and found that the algorithm above gave the results that fit their understanding best of which people are the most active participants on the lists. The next step we need to take, is to figure out the minimum amount of participation we would expect from someone belonging to the group of marginal people. My informants identify that the top 150-200 participants belong to the category they would describe as active community members (employees and non-employees). In order to be within this group an index-level of 30 is needed. It seems therefore that a reasonable index level could be set at 30, and would indicate noticeable participation on the lists. Sorting the list with the algorithm above and an index level above 30 yields the following results:

In total there are 174 users with index  $> 30$  from this time period. Of these *40 users are Novell-employees*. Below is a chart where these *employees* are distributed according to their participation-rank in five equal categories (intervals of 40):





*Figure 17. Participation by employees on mailing lists.*

The chart in figure 17 may need a little explanation: Imagine that we were looking at the list of the 174 participants with index level above 30, ranked from the most active on the top to the least active at the bottom (the ones on the bottom will have an index level close to 30). The chart shows us that among the 40 most active people at the top of the list, there are 15 employees (the user ranked as number one on the entire list is in fact an employee). Among the next group from 41 to 80 in rank, there are 8 employees, and so forth. As we can see, the numbers drop increasingly in the categories that follow. This would indicate that the chosen index level includes those employees that in fact *are* active in the community, and that we do not need to worry about having excluded many employees that otherwise should have been taken into account. Based on this we can say that the amount of marginal people within Novell amounts to no more than 40 individuals, which is 13% of the employees in Novell's Linux R&D<sup>24</sup>.

The other set of marginal people involves the community members on the other side of the organizational boundary, whom also are active in both systems. It may be more difficult to show how they can take part in Novell's organizational system without being employees. Although they are not able to make any direct decisions in this network, they may be rather influential on decisions with the organizational system as they are trusted with some level of responsibility.

<sup>24</sup> Note, however, that the employee-count in the chart does not differentiate between departments within Novell. Some of them might be from other departments than Novell's Linux R&D.

They may even be treated as if they inhabit positions in the organizational system. A community member I spoke with (interview #25) has taken the responsibility of maintaining a popular package for the desktop interface. Although a package maintainer in Novell is officially responsible for this software, the external community member does most of the work with this specific package. He therefore also has a natural level of influence on matters concerning this project. Other examples of marginal people on “the outside” may include community members in decision-making structures such as the openSUSE board, or members whom are assigned authority to work as editors of the wiki’s news-pages. There are more examples such as these, but they are more the exception than the rule as Novell has released little control of the product to external community members.

The number of external members that are included by this definition is harder to count than in the previous case. We can not simply use the activity on the mailing lists as an indicator, as high communicative participation does not reveal whether you are a trusted and influential individual in the Novell system or not. We may however use data from the contributor-survey:

**5.1. To what degree do you experience an ability to influence and take part in decisions involving the openSUSE project?**

If you would like to answer "I do not know", please leave the question unanswered.

| not at all | to a little degree | to some degree | to a large degree |
|------------|--------------------|----------------|-------------------|
| 19 %       | 31 %               | 37 %           | 13 %              |

N = 222

*Table 8. Community-influence on decisions.*

The question in table 8 above shows that 13% of the respondents experience a large degree of influence on decisions in the project. Since there are 222 respondents on this question, it accounts for 26 individuals<sup>25</sup>. This number seems more reasonable, compared to 135 external community members with index level above 30 on the mailing lists. Remember also that the survey only includes respondents that participated voluntarily and on their own initiative, and that the absolute number of people that experience high level of influence in the community therefore should be somewhat higher.

With the numbers in the table above, we can make an educated guess that there may be approximately 30 members in the community that belong to the group of marginal people. But to what degree can we trust that these numbers in fact represent the people we are after? Do they

<sup>25</sup> Actually, it accounts for 28 individuals. 2 of these are however employees, and I have therefore subtracted them from the number.

really have any influence? To investigate this question I searched for correlation between the respondents that experience a high level of influence with other variables in the survey. The results show that these same individuals also respond positively to being motivated by getting their footprint in the distribution, getting acknowledgement for their work and from the fact that the distribution has an enterprise user-base (questions 3.5, 3.6 and 3.8 in the survey, with Gamma above 0.45 on all cases). More importantly, high levels of influence also correlate with high contribution hours, meaning that contributors who put in more work also have more influence (or vice versa). Another interesting finding is that level of influence correlates moderately with having social ties (Gamma 0.37), and that this correlates again with amount of hours contributed. Put together, it seems that there may exist a group of core contributors having influence on decisions, on the product, and receiving acknowledgement for their work. They have some social ties since they communicate a lot together, and they are motivated by the fact that their work is “out there” among a large user base. These findings therefore offer credibility to the assumption that there exists a handful of marginal people on the external side of Novell’s boundary. This finding is also backed by openSUSE managers in Novell that identify the same people as the external contributors they experience as most active and valuable to the project. Their number balances with the total amount of marginal people on the inside of the company as well. In total, this group of marginal people is not very big in numbers. They nevertheless play an immensely important role in making the model of collaboration between Novell and the openSUSE community work in practice. This is how:

### **The strategies and importance of the marginal people**

The common denominator of the marginal people, whether they are employees or not, is their strong normative commitment to the openSUSE distribution and the joint development model. While the boundary objects and communication channels provide a collaborative infrastructure, they do not fully explain how the systems are held normatively together or how the link is protected from internal and external pressure threatening to break it. Examples of such internal pressure may be the old Novell organization’s resistance to change and reluctance to fully embrace the open source development model, or the openSUSE community’s desire to be an autonomous open source community. Externally, pressure may come from the commercial software market upon Novell to monetize their investments with short-term proprietary solutions. On the other side, the openSUSE community may be under attack from the larger open source community’s lack of faith and criticism towards the project. Underlying the pressure for separation is the normative structure of the open source and commercial software models, contrasting each other. A figure of these examples is provided below.

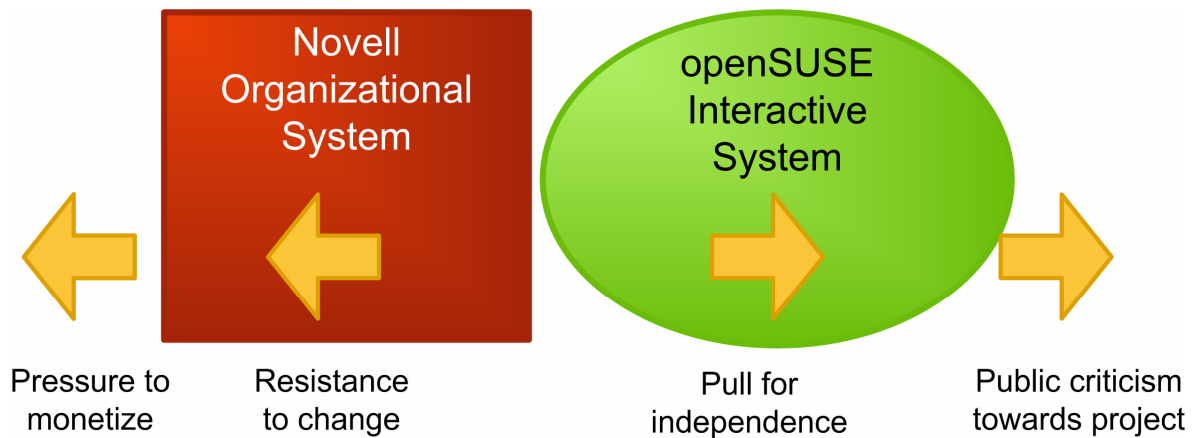


Figure 18. Internal and external pressure towards separation.

Although Novell's top executives and official strategies may provide the structural link between the two systems, it is the work of the collaborating developers that enacts this structure in practice. They represent forces that are important in explaining how the pressure modeled in figure 18 is countered. The practice of the marginal people contributes to the creation of a new social reality, where the hybrid software model (which combines the commercial and open source model) represents the dominating understanding. The strategies they employ (that I have identified) are either:

*enforcing*, such as putting in effort to make the collaborative development model work - and thereby proving its worth;

*aggressive*, such as advocating the model in public discussions and blogs, convincing colleagues to join in, or defending the model against external attacks; or

*symbolic*, as in leading by example and inspiring others to support the project by showing their own commitment.

An example of the latter is provided by this informant, after he was asked what could cause the community to abandon Novell: [W]hat you can do is to fire key people inside the company. For example, if we had fired [John] then other people [in the community] would get really pissed off." (interview #10). Novell have in fact been pursuing this strategy consciously, by recruiting respected and known open source developers to key positions in the company. By holding these positions in Novell, these key people visualize to the outside community that they perceive Novell as a legitimate open source actor themselves. (This does not mean, however, that these individuals do not voice their disagreements with Novell or that they pose as puppets on behalf of the company).

Examples of the aggressive strategy may be found everywhere, on mailing-lists, blogs, in the lunch room or at conferences and expositions. This is a strategy that within the community is commonly referred to as “evangelizing”. Novell even has “evangelist” as an official job-description. Aggressive strategies may happen pro-actively or re-actively. The following is an example of the latter: During the autumn of 2006, Novell signed a deal with Microsoft which created controversy in the open source community. Some claimed the deal would threaten open source developers’ motivation for working with Linux, and Novell in particular. (This issue is disputed, and I will not go into any more detail here). Shortly after the deal was announced, the top manager of an alternative Linux distribution posted a mail to the opensuse-project list, lamenting Novell’s deal with Microsoft and inviting opensuse-developers to join “his” distribution instead. The first reaction that followed was the reply of an external community member who wrote:

“I think the tone of this message is controversial. You can have any opinion you like against the Novell/Microsoft agreement, but taking this to try such hack against openSUSE developers is very far near a war declaration. I hope it's not. For I don't see [other distribution] better than openSUSE on any respect.”

Several more replies followed, rejecting the invitation and defending the integrity of the openSUSE project. Many of the replies were sent by marginal people in the openSUSE community, employees and non-employees alike. The example illustrates the importance of their efforts: this publicly available email-tread must undoubtedly have been read by hundreds of openSUSE community-members, if not more. If the core group had not responded so quickly and decisively, the “attacker” might have succeeded in his mission<sup>26</sup>.

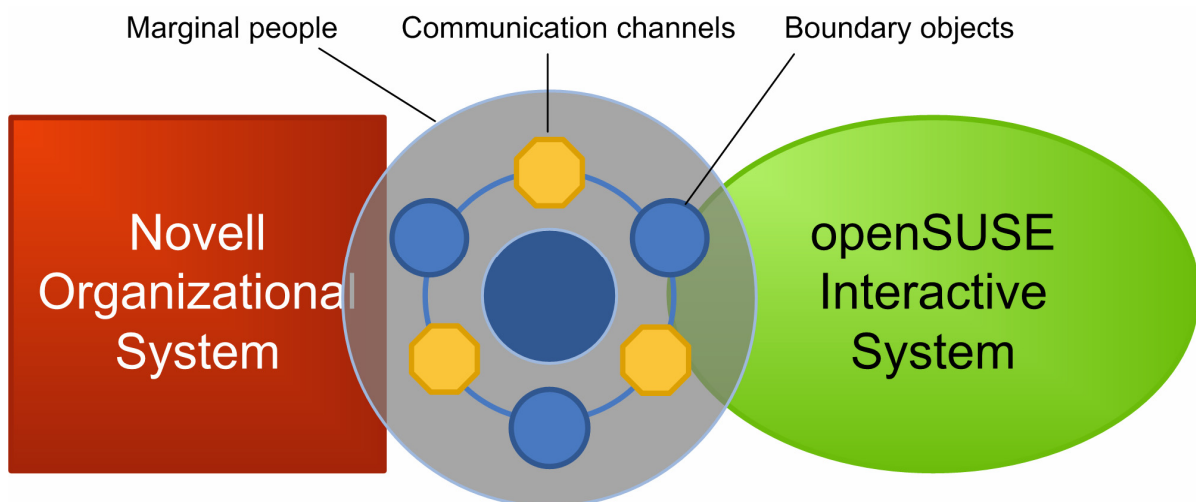


Figure 19. Elements of system connectivity.

<sup>26</sup> I base the assumption that he was rather unsuccessful on information from two openSUSE employees.

In summary, the analysis above has shown how two systems co-exist and collaborate in the creation of a software product. The discussion has focused on explaining how the collaboration is held together by the means of boundary objects, shared communication channels and the effort of marginal people in both systems. The three elements are intricately linked, and work in unison at weaving together the two groups, as shown in figure 19 above.

The underlying assumption is not an understanding of the two systems converging towards an equilibrium, as we can find in functionalist thought. The two systems are in constant reproduction and change, and their relations to each other are by no means static. In the next section we will therefore look at what these theories may tell us about the future development of Novell and the openSUSE project.

### ***Future perspectives***

The Novell organizational system and the interactive openSUSE system are autopoietic and under constant internal reconstruction. They can evolve in several possible directions. Michéle Morner (2003) argues that open source projects provide a good example of autopoietic evolution, as some projects seem to stabilize while others die once communications stop. With openSUSE, anything might happen. Novell, on the other hand, is a somewhat more stable system, but is also constantly evolving and shifting its boundaries between the two systems. It seems as if Novell's evolution is somewhat experimental, and adapts itself to changing conditions and responses from the environment. For example, Novell did not know if opening Bugzilla would be successful, and were prepared to close it again if the results did not turn out as expected. It however proved to be a large success, stimulating Novell to move on with openness in other areas.

Uncertainty exists as to how Novell will solve the dilemma of granting external contributors access to commit code to the code base and strengthen the collaboration with the community. We are standing at a point in time where many things may happen, and the tension between control and openness can drive the relationship between Novell and openSUSE in different directions. At the time of writing, the situation has already evolved and changed since the data was gathered in the autumn of 2007. I will address these changes towards the end of this chapter. In the following, I will present a few possible scenarios for the future evolution of Novell and the openSUSE community that are interesting to discuss empirically and theoretically. We will see that Novell have already started to implement the last of the scenarios that are presented here. However, the evolution of Novell and the openSUSE project is not a given, and a failure in one scenario may lead to another. Moreover, these scenarios could be applicable to any firm-sponsored open source community and can therefore be generalized to a larger population. An overview of the scenarios is provided in table 9 on the next page.

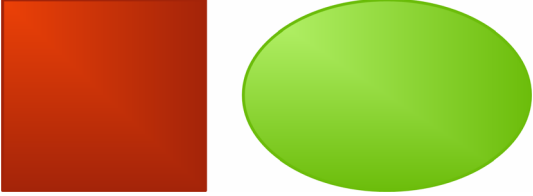
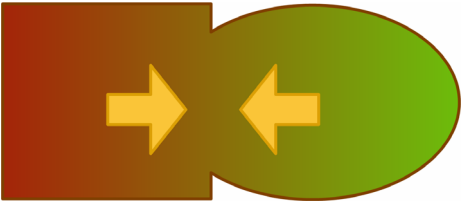
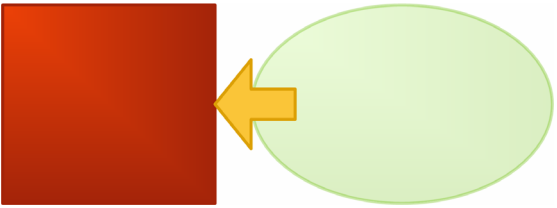

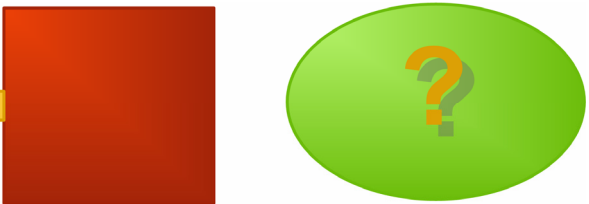
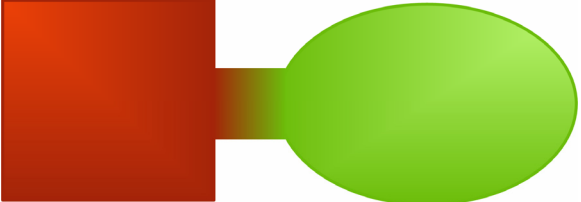
|                                                                                                                                                                                                              |                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <p><b>Status quo:</b><br/>The relationship between Novell and openSUSE remains as it is. Boundary: Continued <i>selective permeability</i>.</p>                                                              |    |
| <p><b>Integration:</b><br/>Novell and openSUSE engulf each other and become one and the same system. Boundary: <i>Complete permeability</i>.</p>                                                             |    |
| <p><b>Extinction:</b><br/>The interactive openSUSE-system dies, and the development goes back into Novell. Boundary: Back to <i>impermeability</i>.</p>                                                      |   |
| <p><b>Divergence:</b><br/>Novell and openSUSE move in opposite directions, and openSUSE evolves into an autonomus organizational system. Boundary: <i>Diverse permeability</i></p>                           |  |
| <p><b>Failure:</b><br/>Novell's business model fails and the company pulls out of the openSUSE project. openSUSE's fate is uncertain and open for anyone to grab. Boundary: <i>Dissolved</i></p>             |  |
| <p><b>Bridge:</b><br/>Novell and the openSUSE system continue to exist as today, but a clear bridge is built between the two systems, enabling mutual influence. Boundary: <i>Qualified permeability</i></p> |  |

Table 9. Future scenarios for the openSUSE project

## Status quo

The first possible scenario is for the current relationship between Novell and the openSUSE system to continue to stay as it is. This would mean that Novell's boundaries will continue to exhibit selective permeability, meaning that all decisions are made by Novell and that any external input is filtered selectively. Basically, Novell can choose if and how they want to use any external contributions, and external developers need to go via employees if they want to commit code into the code base. This situation contributes to the tension between openness and control, but as I have shown in the previous discussion, the potency of mechanisms such as boundary objects and marginal people are strong enough to manage this tension. However, my data indicate that these mechanisms will not suffice indefinitely, rendering this scenario an unlikely permanent situation. Among community members I have been in touch with, the lack of ability to contribute directly is rated as the largest current challenge for the collaboration between Novell and the external community, and must be dealt with soon: "It's critical. Objective number one I would say (...) Because you can't on the one hand say that 'hey, be involved, join and help us!' and on the other hand say 'yeah, you can work on that, but you cannot touch those important things.'" (interview #18 with community member). The issue was also repeatedly mentioned by external contributors in the commentary field in the survey, as this example illustrates: "We need to make it easier for non-Novell people to improve the distribution and have their patches integrated." This pressure is acknowledged by Novell, who also agree that something needs to be done. One of the openSUSE managers claims that "we really need to provide a way to directly leave your footprint there if you want it." (interview #9). A product manager in R&D elaborates:

"What we need is a defined process and rules for how external people can simply contribute to the code. As simple and directly as possible, and at the same time ensure the quality, the security and everything. And that's a tricky task" (interview #8).

So, although the current arrangement may hold for a while, it is likely that the relationship will evolve into one of the other scenarios in time.

## Integration

One option is that the development of openSUSE is opened up completely, and all differences between employees and external developers are erased. In this scenario, we might witness a situation where Novell and openSUSE engulf each other. External developers will be included in the decision network and be embraced by Novell's organizational system. This would not mean that external developers become employees of Novell, but rather that anyone external to the company may gain rights to commit code and part-take in decisions at the same level as employees. Developers might also be given responsibilities as if they held positions within the organization. In O'Mahony's terms, it would mean that Novell's openness moves completely

---



from *transparency* to *accessibility* (2007a). Similarly, Novell’s organizational system will extend beyond its boundaries of the employee-network and into the interactive system, and Novell will increasingly participate in external open source software projects. The company itself would have to shift more of its identity, as the openSUSE distribution will be increasingly *community managed* (O’Mahony, 2007b). I will return to this point in a moment. In the integrated model, the boundaries between the systems would be completely permeable, meaning that traffic and access into and out of Novell’s development process would flow *indiscriminately*. In practice, it would mean that Novell and the openSUSE interactive system become one and the same system<sup>27</sup>. It would also mean that the goals of Novell and the openSUSE system not only remain compatible, but are in fact *harmonized* so they become one and the same. This leads us to our first obstacle.

There are some problematic issues with the integrated model, which might make it an unlikely outcome of the ongoing evolution. First, is it possible to fully integrate Novell’s goals with the aims of the external developers? My data indicate that community members may have a hard time of incorporating the company’s interests into theirs:

[Myself]: “How much do you think the community members identify themselves with Novell, as opposed to openSUSE?”

[openSUSE manager]: “Not so much. I guess there is also this fear about Novell having too much control. And every time Novell does a stupid move, it falls back on to the community. But I don’t think that there are many people in the openSUSE community that identify themselves with Novell” (interview #9).

I investigated this relationship closer in the contributor survey. Here are some results:

#### 4.1. I believe Novell is an open source company

| not at all | to a little degree | To some degree | to a large degree |
|------------|--------------------|----------------|-------------------|
| 12 %       | 19 %               | 50 %           | 19 %              |

N = 273

#### 4.2. openSUSE and Novell are one and the same to me

| not at all | to a little degree | To some degree | to a large degree |
|------------|--------------------|----------------|-------------------|
| 37 %       | 28 %               | 26 %           | 10 %              |

N = 275

<sup>27</sup> Remember that the interactive system is not equivalent to the openSUSE community – there would still be an user-community surrounding the organization in this scenario, but it would make less sense to define external from internal contributors and users.

**4.5. It would be fine with me if openSUSE was designed in red Novell colors**

| not at all | to a little degree | to some degree | to a large degree |
|------------|--------------------|----------------|-------------------|
| 55 %       | 21 %               | 15 %           | 25 %              |

N = 268

*Table 10. Identity between Novell and the openSUSE community*

Although many community members believe Novell is an open source company (at least to some degree [69%]), the results from the next questions in table 10 show that most community members still distance themselves from Novell. The quote and the survey-results therefore illustrate that a majority of the external contributors draw a clear line between the company and the community. This could indicate that it would be a challenge to succeed with harmonizing the two systems. However, these data only illustrate the *current* situation. They do not prove that the situation cannot change, only that it might be a challenging task.

This brings us to the second problem, which concerns the changes Novell would need to make in order for the integrated model to work. A closer examination of the data above indicate that Novell would have to become a “pure” open source company in order to wipe out the differences between the two systems: Statistical analysis shows there is a strong correlation between question 4.1 and 4.2 (Gamma = 0,55), meaning that many of the individuals who see Novell as an open source company are the same that see openSUSE and Novell as one and the same. This would indicate that in order for community members to identify with Novell, the company needs to “be more open source” in order to get the “rest” of the community to feel the same way as the 10% in question 4.2. There are however several reasons why this strategy would be problematic for Novell. Not only does the culture within the entire company need to change and more proprietary products become open, but Novell also needs to release complete control of the product development and let it become entirely community managed. Not all openSUSE engineers believe that Novell will be able to achieve this:

“I don’t see at the moment that Novell will give up complete control of openSUSE. And I am not sure about how this is communicated, and I see this as one of the bigger problems the project will face in the next couple of years. Because at *some point people will begin to expect something*. Because at the moment they are allowed to build packages, and to help other people on the mailinglist and on the wiki, but what they *say* is completely... Well, [a few of the managers] listen to it and we welcome ideas, but if there is a real conflict, for whatever technical thing in the distribution, *Novell will always win*” (interview #23, my emphasis).

The difficulties of implementing an integrated model does not mean that openness can not be achieved. There are alternative scenarios that may achieve accessibility in a different manner. We

will return to discuss these in short while. First, let us see what may happen if more openness is *not* achieved.

### **Extinction**

If we for a moment imagine that Novell do not succeed in finding a solution to achieve some form of accessibility, they will risk losing the developers and contributors that really matter. According to the community members I have spoken with, there seems to be an anticipation within the community that the conflict of community-influence will soon be resolved. Developers currently seem to be waiting patiently for access to be given. If these expectations are not met within due time, Novell risks that the active openSUSE community may fall apart since individual developers might move to other software communities instead. The effect of losing developers due to continued denied accessibility could result in that the openSUSE community becomes viewed as a pure “fanclub” in the larger open source community, which may in return inhibit any further recruitment of developers. Without contributors, the communicative events may stop and the system will die. The distribution will probably continue to be developed in-house in Novell, although it may still have big windows providing transparency into the development. In practice, the boundaries will nevertheless go back to being *impermeable*.

I find this model a rather unlikely outcome of the ongoing evolution, even in the event that no change happens in Novell’s relationship to the community. Luhmann argues that systems stabilize when they contain *communication connectivity* and a *systemic memory* (Morner, 2003), as described previously. Since I have argued that these features are quite present in the openSUSE interactive system, it will not fall apart easily. I also believe that the potent boundary objects I have previously described are able to keep people drawn towards this collaboration and prevent any complete flight, as long as Novell keep the objects open and active. The community could however shrink or diminish over time, without reaching extinction.

In any case, the concern of losing the community (or never succeed in building a powerful community that makes a difference) could go away if Novell grant more access to external developers. Why is it not obvious for them to just do so? As previously mentioned, the SUSE Linux Enterprise product is closely tied to the openSUSE distribution since they share the same code base. The largest problem for Novell with allowing unconditional access to the development process is the company’s ability to guarantee software quality to its enterprise customers, since alterations in the openSUSE code base will directly impact the enterprise distribution. Given this complexity, I believe there are two ways for Novell to provide more openness towards the community. The first option is to pursue a strategy where the code base is split in two, so that any “harm” that may come to the openSUSE code base does not have any direct impact on the enterprise distribution.

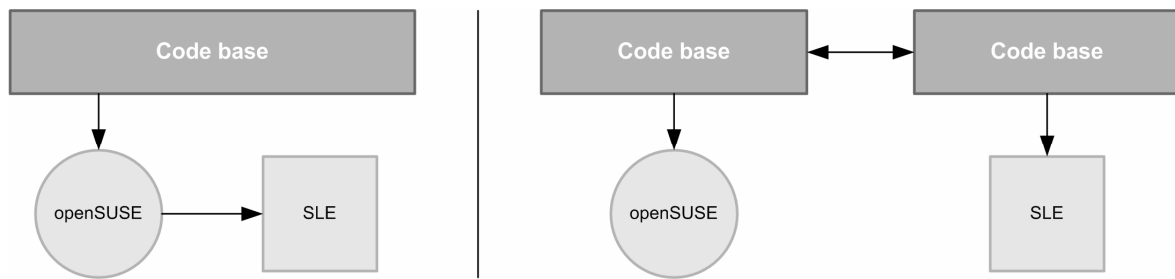


Figure 20. Split code base.

In this case alterations in one of the code bases would only selectively affect the other, as illustrated in figure 20 above. Therefore Novell can take the risk of giving direct access to external developers. According to my informants, this is a model that is used by Red Hat and their open Fedora distribution<sup>28</sup>. Let us have a look at the consequences of this scenario for Novell.

### Divergence

If Novell were to pursue a strategy as described above, it would increase the likelihood of a scenario where Novell and the openSUSE community diverge apart. If the code base is split, the interactive openSUSE system would control its own distribution. This would definitely give rise to an organizational system surrounding the openSUSE object, as a structure needs to be in place in order to handle the decisions that need to be made. Positions and responsibilities within the community will be defined, and decision protocols will be developed. The openSUSE interactive system will thus develop its own decision network and evolve into an organizational system in its own right. In the illustration of this scenario, the openSUSE system is therefore visualized as a square rather than an elliptical. The boundaries of the Novell and openSUSE system will have *diverse permeability*, since there will be different access to the openSUSE development and SUSE Linux Enterprise development in Novell – the former being far more open.

In the event of this scenario, it is likely that Novell and the openSUSE system will drift more apart from each other. This is largely because the openSUSE object will alter its identity and shift its position, and thus lose some of its “power” to draw the Novell organization close. It will no longer be a target-object between the two systems, as Novell engineers will now work directly on the enterprise distribution rather than via openSUSE. The other supportive boundary objects will – instead of being one set of objects between the two systems – become duplicated in two sets and exist as naturalized objects within each of the systems. Their role as “glue” between the systems will therefore disappear. Communication channels will also be more concentrated on connecting the systems internally, and communication across systems will dampen.

<sup>28</sup> In Fedora, voluntary developers are able to achieve commit-rights to the code base through a trust-rating system. I do not have any data on numbers of developers or how successful this model is in their case.

Is this model desirable for Novell and the openSUSE community? This question is not for me to answer. One of my informants claims it will cost Novell a lot of money and recourses that they might not have, as it is more costly to maintain two code bases than simply one. My informant also has a theory that having only a single code base is a strength to the company and the community:

“I think it is an advantage [to have the same code base]. Because we have... Suse linux has some reputation of being made in Germany and have some quality (...) We would give this up [by splitting the code base]. So what people told me, at exhibitions, is that they were happy that we do not give it up completely, that we do NOT do it like Fedora – just buy the brand name and let the community do the work. Because there is quality attached to it, and that we need the code base for the business products. So I think it is an advantage, but I think it is an advantage you have to communicate” (interview #23).

Again, I wanted to find out if my informants' impression was confirmed by the community:

**3.8. I participate in the openSUSE community because my work for openSUSE may be used by many people, including enterprise level**

| not at all | to a little degree | to some degree | To a large degree |
|------------|--------------------|----------------|-------------------|
| 14 %       | 17 %               | 32 %           | 38 %              |

N = 263

The results show that the link to the enterprise product actually motivates a lot of people, but so do many of the other variables in the survey as well. However, the responses to this question correlate moderately with the question of how many hours the members contribute weekly to openSUSE (Gamma = 0,35). This means that for the majority of active developers in the openSUSE community (those who contribute the most), the fact that the product they influence is the same that is distributed through Novell's commercial network is important in explaining their motivation to participate. It would therefore seem as if my informant above has a point. This may indicate that openSUSE *might* lose developers or fail to attract enough talented effort in this scenario, since the openSUSE object can lose one of its important identity characteristics (engineered to high quality and distributed to enterprise users). However, the opposite might also be true. The increased accessibility to the distribution can recruit a large amount of developers to the community. In sum, I can not say whether this scenario is good or bad for Novell or the openSUSE community, but it is certainly an alternative.

## Failure

Another scenario that may occur, and which dawned on me rather late, is that Novell might lose interest in openSUSE and abandon the project altogether. This may happen if the business model fails and if it no longer benefits the company to have a Linux operating system among their products. I could only speculate in the reasons for why this would happen, but it would most likely mean that the firm-sponsored community model failed (although it may be for other reasons than the development-model itself). The consequences for the openSUSE community – which already is a somewhat stabilized system – will depend on the fate of the Novell employees working on openSUSE; whether this part of the company is sold out so they can continue Linux development or directed to work on other tasks in Novell. One manager said that he hopes that “even if Novell would abandon the project, the project would be so lively that it would live on without the corporate sponsor” (08.05.08). Due to this uncertain fate, a question mark is put in the visualization of this scenario. In some form, the openSUSE distribution will continue to exist although the development effort is likely to be severely reduced.

## Bridge

As mentioned previously in this chapter, I believe there are two realistic ways for Novell to provide more openness in their relationship to the community. The first is to split the code base and cut openSUSE loose so that full control of the enterprise distribution can still be maintained by Novell. The second approach is to take steps to scale up Novell’s quality review processes and system of trust so that it can include certain external contributors. This is the final scenario I will present in this section, and the model Novell and the openSUSE community seem to be heading towards already.

In this model, accessibility is given *discriminately*, meaning that contributors have to qualify to earn the trust to gain similar rights as employees towards the code base. I therefore name this boundary *qualified permeability*, as it will permit certain individuals to roam freely between systems within a defined route of passage. This is visualized as a bridge between the two systems in the illustration of this scenario. In a way one could say that Novell would be extending their decision-network into the openSUSE interactive system, but they will still be separated as two systems since external developers will still not have the full rights (and obligations) as employees on other areas. In this scenario, Novell will trust certain external developers the access to submit changes and updates to the code base, and the opportunity to gain this status should be equal to all external contributors. The quality review process should treat internal and external developers in the same manner, and thus ensure that quality is maintained even if the code is received from outside the company.

### The shrinking arm

Interestingly enough, Novell have already been able to move quite a step along the path of this scenario during the writing of this thesis. I constantly noticed new topics and threads concerning this issue on the project mailing lists over the next months after my visit to the company. I was also reminded about this development towards the end of my work. According to our agreement, I sent drafts of my text to Novell for their review, and received comments in return. A director of one of the departments in OPS R&D writes:

"You speak about 'Holding the community at arm's length' - in several places [in the thesis]. I would like to see added that the arm is shrinking ;-)" (07.05.2008).

In a following phone conversation, he adds that his goal is to "erase the distinction between external and internal contributors" (08.05.2008). In May 2008, Novell will release a new version of the openSUSE Build Service, where "internal and external maintainers are handled technically equivalent. This means the Novell employees can use the same accounts and technical solutions as external developers" (written comment from OBS-maintainer, 08.05.2008). In other words, Novell will have made it technically possible for external developers to commit code in the same manner as internal engineers, and integrated the OBS with the Autobuild system. They will also have moved more of the employees' development process into the OBS, thus creating a joint arena for development between external and internal developers. In theoretical terms they are strengthening the use of the boundary object instead of the internal, naturalized object (Autobuild).

When this technical infrastructure is in place, the next step will be to create a system for granting access to trusted developers. How does one decide which individuals are qualified to commit to the code base? And who gets to make the decision? If the two systems are to be maintained as equal partners around the openSUSE distribution, it can not be up to Novell alone to define the qualification criteria or to decide who gets to be qualified contributors. In collaboration with the community, Novell are therefore also developing a *user-trust model* that will be technically integrated in the OBS. The goal of a user-trust model is to have a system that can calculate a trust-level<sup>29</sup> for all registered developers that wish to share software with other users. The point of such a system is based on the following principle: "In general, the trust in a certain software comes from the people behind it. If we can trust them, we can trust their software."<sup>30</sup> The rating of a developer in such a system can depend on a number of factors. A discussion-document on the project includes these working suggestions of factors that will affect developers' trust-level:

---

<sup>29</sup> For example, trust-level can be reported as a number between 1 – 100.

<sup>30</sup> Quoted from one of the working documents on the user-trust project, at [http://en.opensuse.org/Build\\_Service/Concepts/Trust](http://en.opensuse.org/Build_Service/Concepts/Trust)

- users accepts and signs a contract
- rating from other users
- user registers personal contact details
- user obliges to deliver updates for software project (ibid)

By defining criteria such as these, the community is also creating a system of incentives for stimulating developers to pursue trustworthy activities (in order to gain a higher ranking). When the user-trust model is in effect, every user that wants to download software from the OBS can see the rating of the author of the project and thus get an indicator of whether the software can be trusted. More importantly (at least for this discussion), the trust-rating system will also provide an instrument for Novell to qualify external developers, by using a measurement-tool and criteria that have legitimacy in the external openSUSE community. Once the qualified developers start submitting contributions towards the common code base, Novell's quality review processes will ensure that the contributions also meet the necessary standards, in the same manner as all contributions from employees are scrutinized.

In sum, Novell are constructing the bridge between the two systems without the need to drastically change the company or eat up the community. This technical and social bridge will be an important mechanism for managing the tension between openness and control between the Novell organizational system and the interactive openSUSE system. It is built by *empowering the objects* situated between them, especially by strengthening the role of openSUSE Build Service and giving externals more access to the openSUSE object. The second important consequence of pursuing this strategy is that it may *expand the group of marginal people*, on both sides. Employees that previously strictly did their development indoors through Autobuild, will expose their work in the Build Service and are thus likely to interact more with the external community. Likewise, external developers may achieve rights to commit code through the OBS, and thus be part of Novell's decision-network. I believe this may be a promising strategy for *managing openness*. But, although the structure of this scenario is falling into place for Novell and the openSUSE community, it however remains to be enacted in practice.



# Conclusion

---

The case of Novell and the openSUSE project has been an interesting study, and has provided a great insight into the relatively new domain of firm-sponsored communities. Approaching the concluding remarks of this thesis, it serves to ask: What have we learned? How can we use this knowledge and take it further?

### **Luhmann, Star and Novell**

My first research question sought to investigate the nature of the firm (i.e. Novell) in relation to the external community. I have found that different understandings of this relationship are expressed by my informants. For example, within Novell's US-based marketing department, there is an assumption that the openSUSE community is an independent network of individuals that control the openSUSE project: "We don't own it. (...) So they [the community] kind of take the project in the direction they want to" (interview #5, 25.5.07). This description is not very accurate. A quick glance at the characteristics of software projects that are entirely community-managed (O'Mahony, 2007b) will reveal that the openSUSE project does not fit the case of an autonomous organization, largely because the external community has had no decision-making rights or way to directly influence the development of the product. A second view I have encountered of Novell's new organizational construction is to see the company and the community as entirely integrated; meaning that the external community is *part of* Novell and that Novell therefore is a "pure" open source company. In my analysis, I find that neither of these understandings above are true. I have found that Niklas Luhmann's theory of autopoietic social systems is precise at capturing the differences between Novell and the external openSUSE community, as it defines these two as different types of social systems; an organizational and an

interactive system. Luhmann's minimalist understanding of social systems as being based on communicative events is also helpful in identifying the boundaries of the active parts of the openSUSE community, which my informants otherwise find quite difficult to frame. To be part of this interactive system, no contracts are needed – only direct participation. This understanding fits the open source methodology well, and I therefore believe autopoietic systems theory is especially useful in this context. I also believe this analytical model can be generalized to the larger population of firm-sponsored communities, as they are likely to share similar characteristics as Novell and the openSUSE community in terms of dilemmas of decision-making and control. In addition, the emphasis on process in Luhmann's theory provides a historical dimension that I believe is important to capture; if a developer contributed a patch to openSUSE once several months ago without having participated since, he would probably not identify himself as part of the active community. However, this action is part of the systems history; it is a communicative event that has had an impact on later communications in the system and is thus part of what explains the current status of the system. To return to the answer of my first research question, I have found that the openSUSE community is distinguished as an interactive system from Novell's decision-network. The openSUSE community is not an organizational system in its own right, but has features that nevertheless make it a fairly stable system that has a boundary towards Novell in the development of the openSUSE distribution.

The second research question addressed the *connectivity* between these two systems: How can they, despite their conflicts and differences, manage to stay united around the development of the openSUSE distribution? I have emphasized the role of boundary objects, shared communications channels and marginal people in my response to this question. The insights from previous research on boundary objects show how such objects can be supportive of multiple, simultaneous translations of interests. This knowledge has been particularly useful for explaining how potential difficulties in the collaboration between the organizational and interactive system are overcome. The solution has not been to battle disagreements on opinions and values, but provide room for all of them to co-exist. If a developer does not "win" in a development-dispute, the openSUSE Build Service will allow him to take things in his own direction, rally support and prove the others wrong.

The objects are also important in explaining how the systems are "glued" together, at the same time as the objects maintain the distinction between the systems. The factory code base is the most prominent example of an object embodying this duality, as it receives focus and attention from all developers. It represents the latest developments and manifestation of decisions, yet at the same time it provides different access-levels to employees and external developers, thereby differentiating them. The strongest attraction exists in the openSUSE object, that unites the interests of all actors. It is a shared, collective object that has meaning for each group and each

---

individual. In explaining how the systems are held together, I have also emphasized the importance of marginal people whom actively part-take in both of them and whom are the most vigorous users of the shared communication channels.

### **Exploring theory**

My third research question has directed me towards pushing the limits of the theoretical framework I have applied in this thesis, by trying to investigate what the data from my case may do for these theories. I will start off this theoretical summary by accounting for some of the unintended insights that I have stumbled over along the way of this research project. They have been useful for me and might be helpful for others as well. For one, it has been interesting to experiment with metaphors from the natural sciences, and I have found it useful to develop a terminology for describing the nature of organizational boundaries in terms of their level of *permeability*<sup>31</sup>. As more and more product development is following models of open innovation, such a vocabulary may come in handy in describing similar cases as the one of Novell. Second, I have found that using an iterative and dynamic research design that can be changed along the way, is not restricted only to qualitative methodology. The quantitative measures I initiated for this research project were spurred by the moment, and gave me new answers to further my qualitative research. With more time, I might have initiated other surveys or statistical investigations. I must note, however, that my quantitative escapades were quite small in stature, and may of course not be as easy to initiate in on a larger scale. But, I do believe that the evolution of information technology has made such measures easier to execute. Electronic survey-technology exists at most universities and more and more information is available for easy access to analyze on-line. I therefore think researchers have an opportunity to pursue these advantages further to create more dynamic quantitative research designs.

To address the theories that I have *intended* to explore, I will start by saying that not only has the case of Novell, in my opinion, done justice to Luhmann's and Star's theories by showing the value of their application, but the case has also brought some new and additional insights to these theories. First of all, Novell's case shows how well they (the theories) can work together. I mentioned earlier that I have failed to find elements in Luhmann's theory that adequately describe how social systems may be interrelated. I find that boundary objects are a promising supplement in this respect, and the case shows that these kinds of objects also may be situated between *systems*, as well as individuals and groups.

This thesis has aimed at exploring Star's objects in depth. At an early stage of the analysis I was mildly annoyed by the fact that this theoretical construction – which I found very useful – could

---

<sup>31</sup> I extend my thanks to Lars Risan for this idea, and whom deserves credit for several discoveries formulated in this thesis.

not differentiate between the position and role of the objects held in the collaboration, which ultimately led to my distinction of supportive- and target-objects. I have argued that the latter hold a strong motivating power that provides an explanation to why collaborators come together in the first place and how they are held together at a higher level than the social mechanics of translating compatible interests. To illustrate this point, I have drawn upon Emile Durkheim's theory of organic solidarity to explain how social separation – that may seemingly appear as separation of social worlds – in fact is a separation *within* the same world, and that Durkheim's notion of collective symbols is important for explaining how this internally diverse world stays together despite the separation of its consistent parts. I have also appended our knowledge of epistemic objects to argue that these symbolic target-objects also have a functional purpose that provide further motivating qualities.

Although I have a list of theoretical features I have wanted to add to Star's boundary objects, I do not criticize her theory of being inconsistent. The boundary object concept does not aim to explain "why" people want to collaborate or how the collaboration comes into being. It only describes how they actually do it. However, I believe Star's theory should not settle with this lack of ambition, *because I do not find that the motivation for collaboration can be separated from the objects themselves*. Since the construction of target-objects is derived from the case in this thesis, it would be interesting to test its value in future research in other contexts. It would particularly be interesting to use this construction to research cases where multiple target-objects exist. Would they contribute to dividing the collaborators, by attracting followers to each of them, or would they work in unison at uniting them?

In developing the theory of boundary objects, Star pointed to Robert Parker's work on marginal people, which has also been very useful for this case. However, I have used this concept in a different sense than in Parker's original work. Another theoretical contribution from this thesis may therefore be the illustration of how marginal people may represent an abundance of *resources* in collaboration. While earlier research might have put more emphasis on the inherent dilemmas that people with dual identities might have – belonging everywhere yet nowhere – these people may prove to be extremely valuable when the two worlds they inhabit are attempted to be united. My research has shown some strategies they have applied towards the external environment. Further research on this area could focus on the role they play internally, as translators, advocates, mediators, diplomats or maybe as entrepreneurs. While marginal people is a notion attributed to people where rather permanent characteristics such as race create this identity, perhaps the concept of marginality may be used to represent more fleeting characteristics, and determine roles in collaboration that represent a property that may shift when individuals move between different contexts of collaboration. Again, this could be an interesting direction for more studies.

Returning to Luhmann, I believe the discussion in this thesis shows that it is possible to complement this theory successfully with elements drawn from other theorists' work. The theory of autopoietic social systems has several interesting features, yet it is also limited in its explanatory power on several areas. To name a couple, both the absence of people and normative rationality within Luhmann's systems can make the theory fall short on important areas. However, I do not claim that any theory could be combined with Luhmann's without scrutiny. It is certainly possible to criticize the attempt in this thesis to merge two theories, particularly because they come from quite different trains of thought. I nevertheless believe they complement each other successfully and may not even be as different as one might initially think. Star's theory of boundary objects springs from a position closely tied to Latour's actor-network theory, which in itself is a form of systems theory (Bakken & Hernes, 2003). In addition, they both operate at a meso-level of analysis, including both individual actions and events, and more structural macro-phenomena. I believe that further experiments with other constellations with Luhmann's theory by organizational theorists could provide interesting new insights.

### **Managing openness**

In summary, what can the analysis and discussion in this thesis tell us about the problem of managing the tension between openness and control between the sponsor firm and the sponsored community? The discussion shows that it is possible to maintain and develop both control and openness, as these properties are organized through two different kinds of autopoietic systems (the organizational and interactive system). Furthermore, these systems are linked through a boundary, or to be more precise, a set of boundary objects. The objects allow for control to be maintained while they also provide a great deal of freedom and opportunities for external development contributions. The general lesson we may learn from these insights is that the chances for success of the firm-sponsored community model are strengthened when: (1) activity and focus shifts from internal, naturalized object to boundary objects; (2) transparency from the sponsor firm is followed by encouraged participation from the sponsored community on shared channels; (3) the group of marginal people is continuously expanded from both sides – by encouraging employees to communicate externally and by increasingly including external developers in the organizational decision-network; and (4) every possible strategy that may make the common target-object an individual property of everyone is pursued, through mechanisms of contribution that supports shared ownership. The latter may unleash the target-object's power to motivate and draw collaborators tighter together.

Novell's transparent form of development – through using the open communication channels and encouraging input and discussion – has been very important for relieving some of the existing tension in the collaboration surrounding the openSUSE distribution. As I have shown in my discussion of Novell's future perspectives, the pressure may nevertheless build up over time if

the situation does not continue to evolve in a direction *towards more openness*. Novell are aware of this fact and are already taking measures to ensure that the arm (that is holding the community at arms-length) is “shrinking”. *Does this mean, then, that the firm-sponsored open source community-model must evolve towards more openness (and thus abandon control) to survive over time?* My data seem to indicate so, in this particular case. However, Novell’s clever maneuvering and their remarkable technical engineering of the Build Service technology shows that there are ways of devolving control in a “controlled” manner, meaning that the openness can be *managed* and that the risk is therefore minimized. Further research on the topic of firm-sponsored communities could therefore aim at investigating if it is possible to control openness, and whether this is the solution for managing communities such as these over time. If successful, this model may prove that facilitating open innovation – by drawing upon the best of both the commercial and open source software models – is possible.

## References

- Bakken, T., & Hernes, T. (2003). *Autopoietic organization theory: drawing on Niklas Luhmann's social systems perspective*. Oslo: Abstrakt.
- Barth, F. (1969). *Ethnic groups and boundaries: the social organization of culture difference*. Bergen: Universitetsforlaget.
- Bonaccorsi, A., Lorenzi, D., Merito, M., & Rossi, C. (2007). *Firms' Participation in Open Source Projects: Empirical Evidence and Research Agenda*: SSRN.
- Bowker, G. C., & Star, S. L. (1999). *Sorting things out: classification and its consequences*. Cambridge, Mass.: MIT Press.
- Brans, M., & Roszbach, S. (1997). The Autopoiesis of Administrative Systems: Niklas Luhmann on Public Administration and Public Policy. *Public Administration*, 75(3), 417-439.
- Brown, J. S., & Duguid, P. (1991). Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning, and Innovation. *Organization Science Special Issue: Organizational Learning: Papers in Honor of (and by) James G. March (1991)*, 2(1), 40-57.
- Bryman, A. (1988). *Quantity and quality in social research*. London: Unwin Hyman.
- Callon, M. (1986). *Some elements of a sociology of translation: domestication of the scallops and fishermen of St. Brieuc Bay*. In Law, John (1986).
- Carlile, P. R. (2002). A pragmatic view of knowledge and boundaries: Boundary objects in new product development. *Organization science*, 13(4), 442.
- Carlile, P. R. (2004). Transferring, Translating, and Transforming: An Integrative Framework for Managing Knowledge Across Boundaries. *Organization science*, 15(5), 555.
- Collins, R. (2004). *Interaction ritual chains*. Princeton, N.J.: Princeton University Press.
- Dahlander, L. (2004). *Appropriating the Commons: Firms in Open Source Software*. Gothenburg: Chalmers University of Technology.
- Dahlander, L., & Magnusson, M. G. (2005). Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy*, 34(4), 481-493.
- Demil, B., & Lecocq, X. (2006). Neither Market nor Hierarchy nor Network: The Emergence of Bazaar Governance. *Organization Studies*, 27(10), 1447-1466.
- Durkheim, É. (1965 [1912]). *The elementary forms of the religious life*. New York: The Free Press.
- Giddens, A. (1984). *The constitution of society: outline of the theory of structuration*. Cambridge: Polity Press.
- Glaser, B. G., & Strauss, A. L. (1968). *The discovery of grounded theory: strategies for qualitative research*. London: Weidenfeld and Nicolson.
- Groth, L. (1997). *Building organizations with information technology: opportunities and constraints in the search for new organizational forms*. Norges handelshøyskole, Bergen.
- Groth, L. (1999). *Future organizational design: the scope for the IT-based enterprise*. Chichester: Wiley.

- Hippel, E. v., & Krogh, G. v. (2003). Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization science*, 14(2), 209-223.
- Holter, H., & Kalleberg, R. (1996). *Kvalitative metoder i samfunnsforskning*. Oslo: Universitetsforl.
- Hughes, J. A., Martin, P. J., & Sharrock, W. W. (2003). *Understanding classical sociology: Marx, Weber, Durkheim*. London: Sage.
- Jönhill, J. I. (2003). *Communications with decisions as medium and form - some notes on Niklas Luhmann's theory of organization*. In Bakken and Hernes (2003): Chapter 2.
- Kirk, J., & Miller, M. L. (1986). *Reliability and validity in qualitative research*. Beverly Hills, Calif.: Sage Publications.
- Knorr Cetina, K. (1997). Sociality with Objects. Social Relations in Postsocial Knowledge Societies. *Theory, Culture and Society*, 14(4), 1-33.
- Knorr Cetina, K. (2001). *Objectual practice*: In Schatzki T, Knorr Cetina K, Savigny EV (eds) "The Practice Turn in Contemporary Theory". London; Routledge.
- Lamont, M. (2001). Culture and Identity. In Turner, J (2001): Chapter 9.
- Latour, B. (1988). *The pasteurization of France*. Cambridge, Mass.: Harvard University Press.
- Law, J., & Hassard, J. (1999). *Actor network theory and after*. Oxford: Blackwell.
- Lee, A. S., Liebenau, J., & DeGross, J. I. (1997). *Information systems and qualitative resarch: Proceedings of the IFIP TC8 WG 8.2 International Conference on Information Systems and Qualitative Research, 31st May-3rd June 1997 Philadelphia, Pennsylvania, USA*. London: Chapman & Hall.
- Lerner, J., & Tirole, J. (2002). Some Simple Economics of Open Source. *Journal of Industrial Economics*, 50, 197-234.
- Lin, Y. (2006). Hybrid innovation: The dynamics of collaboration between the FLOSS community and corporations. *Knowledge, Technology, and Policy*, 18(4), 86-100.
- Luhmann, N. (1988). *Organization*. In Bakken and Hernes (2003): Chapter 3.
- Luhmann, N. (2000). *Organisation und Entscheidung*. Wiesbaden: Westdeutscher Verlag.
- Mathur, P. (2005). Neither Cited nor Foundational: Niklas Luhmann's Ecological Communication; A Critical Exegesis and Some Theoretical Suggestions for the Future of a Field. *The Communication Review*, 8, 329-362.
- Mingers, J. (2003). *Observing organizations: An evaluation of Luhmann's organization theory*. In Bakken and Hernes (2003): Chapter 6.
- Mintzberg, H. (1983). *Structure in fives: designing effective organizations*. Englewood Cliffs, N.J.: Prentice-Hall.
- Mintzberg, H. (1989). *Mintzberg on management: inside our strange world of organizations*. New York: Free Press.
- Morner, M. (2003). *The emergence of open-source software projects: How to stabilize self-organizing processes in emergent systems*. In Bakken and Hernes (2003): Chapter 12.
- Nicolini, D., Mengis, J., & Swan, J. (2007). *Organizing scientific work through objects. A case from the field of bioengineering*. Paper presented at the Society for Social Studies of Science Annual Meeting.
- O'Mahony, S. (2003). Guarding the commons: how community managed software projects protect their work. *Research Policy*, 32(7), 1179-1198.
- O'Mahony, S. (2007a). *Sponsored Open Source Communities: a Vehicle for Open Innovation?* Paper presented at the European Academy of Management 2007.



- O'Mahony, S. (2007b). The governance of open source initiatives: what does it mean to be community managed? *Journal of Management and Governance*, 11(2), 139-150.
- Raymond, E. S. (2001). *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. Beijing: O'Reilly.
- Rheinberger, H. J. (1997). Object Construction and Networks in Research Work: The Case of Research on Cellulose-Degrading Enzymes. *Philosophy of Science*, 64(Supplement. Proceedings of the 1996 Biennial Meetings of the Philosophy of Science Association. Part II: Symposia Papers), 245-S254.
- Ringdal, K. (2001). *Enhet og mangfold: samfunnsvitenskapelig forskning og kvantitativ metode*. Bergen: Fagbokforl.
- Risan, L. (Forthcoming). Selfing Debian and Othering Ubuntu – differences between Whitehead and ANT, and their consequences for how we understand free software. Oslo.
- Seale, C. (1999). *The quality of qualitative research*. London: Sage.
- Seidl, D. (2005a). *Organisational identity and self-transformation: an autopoietic perspective*. Aldershot: Ashgate.
- Seidl, D., & Becker, K. H. (2005b). *Niklas Luhmann and organization studies*. Malmö: Liber.
- Silverman, D. (2005). *Doing qualitative research: a practical handbook*. London: Sage.
- Stallman, R., & Gay, J. (2002). *Free software, free society: selected essays of Richard M. Stallman*. Boston, Mass.: GNU Press.
- Star, S. L., & Griesmer, J. R. (1989). Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, 19(3), 387-420.
- Strauss, A. L. (1978). A Social World Perspective. *Studies in Symbolic Interaction*, 1 119-128.
- Turner, J. H. (2001). *Handbook of sociological theory*. Dordrecht: Kluwer.
- Viskovatoff, A. (1999). Foundations of Niklas Luhmann's Theory of Social Systems. *Philosophy of the Social Sciences*, 29(4), 481-516.
- Weber, S. (2004). *The success of open source*. Cambridge, Mass.: Harvard University Press.
- Wenger, E. (1998). *Communities of practice: learning, meaning, and identity*. Cambridge: Cambridge University Press.
- West, J. (2003). How open is open enough?: Melding proprietary and open source platform strategies. *Research Policy*, 32(7), 1259-1285.
- West, J., & Gallagher, S. (2004). *Key Challenges of Open Innovation: Lessons from Open Source Software*. Working paper: May 2004.
- West, J., & O'Mahony, S. (2005). *Contrasting Community Building in Sponsored and Community Founded Open Source Projects*. Paper presented at the System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on.
- Williamson, O. E. (1987). *The economic institutions of capitalism: firms, markets, relational contracting*. New York: Free Press.
- Woodward, J. (1958). *Management and technology*. London.