Version Control Software in the Open Source Process: A Performative View of Learning and Organizing in the Linux Collectif

Maha Iftikhar Ahmed Shaikh

Information Systems and Innovation Group
Department of Management
London School of Economics and Political Science
University of London

Thesis submitted in fulfilment
for the degree of Doctor of Philosophy
January 2007

F
8867

To mama and papa

Abstract

This research describes a study of learning and organizing within the Linux kernel open source collective. For its empirical focus it concentrates on Linux kernel development activities and this collective's debates about the role of, and need for, an agreed approach to version control software. This is studied over a period of eight years from 1995-2003. A textual analysis of messages in the Linux Kernel mailing list is used as the primary data source, supported by other contemporary accounts. In this work learning and organizing are understood to be mutually constitutive, where one entails and enables the other. Learning is about interacting with the environment, organizing is about reflecting this in the collective.

The thesis uses the theoretical approach of actor network theory, Bateson's levels of learning and Weick's concept of organizing, to analyze learning and organizing in the kernel collective. The analysis focuses on the discourse and interplay between relevant actors (human and non-human), and the ongoing debates among kernel developers over whether to use version control software, and then which version control software to adopt. The persistence and passion of this debate (it spans the 8 years studied and is ongoing) is evident, and allows a longitudinal study of the *becoming* of learning and organizing. Drawing on actor network theory, the thesis emphasizes the performative (worked out, lived, 'in the doing of', in other words the becoming) character of learning and organizing.

The findings of the study reveal how learning is understood in the collective and is, to a degree, reflected in its organizing activity. Key themes that emerge include: the organizing of time and space, maintaining of transparency and the overall concern with sustaining the assemblage. The thesis offers a distinctive account of technical actors as an essential part of the open source process. In conclusion, it re-emphasizes the significance of code and the agency of non-human actors.

Acknowledgements

It is impossible to thank all the actors (both human and non-human) who have supported me intellectually and emotionally in the last few years but I will try anyway. Mama and Papa made it possible for me to be at the LSE – THANK YOU! Tony Cornford, my supervisor, has got to be the best supervisor ever. His amazing patience, good humour and real nurturing spirit has made this thesis possible. Tony, you really are super and I hope we continue to work together in the future. Farooqi, one of my closest pals, has born the brunt of my horrid moods and uncomplainingly (almost) helped me through this process while doing his own PhD. I would like to thank all my family, especially Chanda and Salman, who provided welcome relief from the stresses of a PhD.

I would like to pay my respects to Claudio Ciborra whose amazing ability to refine ideas made me see possible trajectories that my work could take if only I allowed it to speak to me. Sarah Emery, a friend and manager, created the supportive environment for me to both work and continue with my PhD simultaneously. She is just lovely! Susan Scott is a very giving academic and I will always be grateful for her feedback and interest in my work, and the financial support being her research assistant provided to me. Professor Angell, with many a venti latte from Starbucks, and Nietzsche always cheered me up ☺.

Collaboration with Antonio Cordella developed my understanding of actor network theory. Steve Smithson, Carsten Sorensen and Edgar Whitley have helped in so many ways and I would also like to thank Shirin Madon and Erica Wagner for their kind encouragement. Thank you Leslie Willcocks for hurrying my thesis over the finish line!

Linda Trautman, you have been a real darling for keeping my spirits high throughout my PhD. You made me feel that I could finish. I would also like to thank all my LSE friends and colleagues for their support and interesting debates. They include Angela Lin, Adel Al-Taitoon, Federico Iannacci, Murat Baygeldi, Evi Kopanaki, Dionysis Demetis, Cameron Lawrence, Carol Hsu, Matthew Smith, Ashu Khanna, Pro Tsiavos, Frederick Wamala, Yingqin Zheng, Mary Darking and Danny Osei-Joehene.

Last but not least my friends, Salma, Sophie, Azim, Usman and Shamyl, who have shown such fortitude. I have many years of catching up to do with all of you! I hope it isn't too late…

Table of Contents

List of Figures

List of Tables

List of Acronyms

ANT	Actor Network Theory
BB	BitBucket
BK	BitKeeper
CoP	Communities of Practice
CVS	Concurrent Version Systems
GNU	Gnu's Not Unix
GPL	General Public License
LKML	Linux Kernel Mailing List
OPP	Obligatory Passage Point
SCM	Source Control Management
VCS	Version Control System

Chapter 1
Introduction

It's not the answer but the question
that drives us mad...[1]

This thesis considers the question of how a particular technical actor (version control software) is implicated in activities of learning and organizing within an open source project. The work is described as taking a performative view because the emphasis is on what happens or is happening (process), rather than what is achieved or some end state (structure). There is always a balance to be struck between structure and process but this work is biased towards process. Cooper (1976) defines structure as the "invariant pattern of relationships among functional points in a system, while process is the continuous emergence of new elements from those already existing". Structure concerns itself with stability or quasi-stability; process, with change. Drawing on this distinction we seek to reveal change in an open source collective in terms of learning and organizing, in this work taken as an ensemble construct and inextricably linked.

The theoretical perspective used here allows us to construct a framework for analysis that allows us to shed new light on the open source process and how, through the combined agency of people and other artefacts, interests are aligned and new capacities are imagined, and operationalised, what Callon and Law (1995) call an emergent effect. In this work, drawing as it does strongly on actor-network theory (ANT) and specifically the usage of Callon and Law, open source activity is characterised as a *collectif* (or in more standard English a collective) where "a *collectif* is an emergent effect created by the interaction of the heterogeneous parts that make it up" (Callon and Law 1995, p485). In conventional ANT terms, we may see this as a socio-technical account, but following Latour (2005b) we may be bolder and state it as just a social account.

This does not mean that people alone are the focus, but that we are concerned with the society of all things. Thus a social understanding of a collectif lays emphasis on tracing all manner of associations. Social does not imply some privileged kind of material that is social, rather it is a "type of connection between things that are themselves not social" – after all a person alone is not performing socially (Latour 2005b, p5). This claim links with the reason why the word collectif has been adopted here over the more common designation of open source as a community or group. Society and other such words are used so often in everyday language that they too have come to mean some type of material stability rather than emphasising the process of assembling which is implied in the collecting of a collectif. Thus, a collectif "will designate the project of

[1] The Matrix (1999) – a film directed by the Wachowski brothers: Conversation snippet between Trinity and Neo.

assembling new entities not yet gathered together and which, for this reason, clearly appear as being not made of social stuff" (Latour 2005b, p75). The idea of a collective and its binding and evolving force is explained well by Weick (2001) as "collective structures form when self-sufficiency proves problematic" (pp17) so others must be enticed to participate and help achieve goals. He adds that "people commit to and coordinate instrumental acts (means) before they worry about shared goals. But shared goals do emerge as people search for reasons that justify the earlier interdependent means to which they have become bound" (pp17). Weick thus uses his idea of sense-making in hindsight to explain why collectives are created and why individuals collaborate. Learning and understanding happen in hindsight, learning is thus the reflective time *after* an action has been taken.

Learning in the Context of Information Systems

We take a moment here to take a step back and place this work on learning and organizing in the broader context of information systems studies. Boland *et al.*(1994) discuss how computer-based systems should be developed to support distributed cognition. It is the understanding of this researcher that version control systems are one such manifestation of distributed cognition. Huber (1991), talks of information and knowledge synonymously, and as Jones (1995) points out, the work on Artificial Intelligence suggests that "learning may be instantiated not just in information systems, but in information technology itself". Galbraith (1977) and Pentland (1992) both make similar claims about how cognition and the cognitive approach is a "natural outgrowth of the information-processing model of organizations" (Galbraith, 1977).

Jones (1995) provides three ways to link organizational learning with information systems when organizational learning is something that occurs through individuals. Stata (1989, cited in Jones (1995)) provides one such route and argues that information is the same as learning because information is "essential to the learning process" and thus information systems should be designed around the needs of organizational learning and with the latter as the goal. Another way to understand the link is to look for the contribution information systems make to organizational learning, for example the use of information systems to provide an organizational memory. Argyris and Schon (1978) and their idea of theories-in-use is one such example, through which to link with open source and the use of version control software as a repository of code. And finally Jones (1995) adds that information systems also provide mechanisms to support formal and informal learning, like electronic mail to spread stories that create a sharing of organizational experience (Brown and Duguid, 1991). We know that in our domain of study, open source, one of the main forms of communication is via electronic mail.

Domain of Study

Over the last few years open source software development has aroused a growing interest in academic and business communities alike. There is a perceived need to make sense of this seemingly strange and paradoxical phenomenon. What motivates individuals to give their work away freely to others? Why would any business want to release the source code of its product which could then be appropriated and copied by rivals? These and other pertinent questions have absorbed researchers from many disciplines including software engineering, social psychology, sociology, political economy and business strategy. This research is an attempt to understand a specific, and we argue insufficiently studied question, that of how learning and organizing occurs and is manifested in OS activity. The specific focus of the empirical work is to comprehend learning and organizing in OS through a study of the adoption and use of version control software.

A review of the technology used by the Linux developers initially helped to concentrate the work to find a focus on a specific problematic issue, that of which version control software the Linux developers would adopt, but later lead to the realization that version control software is itself one of the strong actors in the network and (potentially) can become the medium [obligatory passage point] through which the collective organizes and developers learn. Oops! Now we have just split the technology from the human and this is not in keeping with the basic premise of actor network theory, that of the social and symmetry. However, as will be seen in Chapter 2, we are often forced to start at least by theorizing the technology apart from the human actors because that is how other research in this area has been conducted. Thus, in order to survey the literature on open source and version control software we have acknowledged the distinction. However, we do see it as artificial and we only adopt it so that we can make transparent the *lack* of symmetry in other work. Then, as can be seen later in chapter 7 we draw the two rightfully together to theorise the collective as truly a hybrid.

It is not surprising that academics with a software engineering background have shown interest in open source as a software development process, but what is perhaps more surprising is that the wider interest in open source, and here we refer to sociologists, political scientists, management etc has to date often concentrated on software development without noticing the software embodied in the process. There has been little attempt to unpack the software or technology to signify anything more than a necessary facilitator. Even Weber (2004, p118) whose ideas we draw on extensively in this work, when discussing the Linux case, states that "technology is a facilitator. For this community in particular, the idea of a technological fix helped bring the parties together on common ground, but ultimately the solution was going to be a political one". What Weber, amongst others, seems to miss is that technology or any software is *itself* a political artefact. It doesn't materialize out of nothing, indeed the (political) effort it takes to be created and moulded is often monumental. But more importantly it behaves (*performs*) as a political artefact (Winner 1986).

This study then focuses on how a political technological artefact (with its own agency), an *assemblage* like version control software, seeks to persuade other different actors to do its bidding or join its programme and perpetuate its use and existence. We can then ask the question, what do such political actors, within the social hybrid mean for the Linux collective and how it organizes and learns, or more exactly the question that this research addresses is - *how does learning and organizing occur and manifest itself in open source collectives?* This is the goal of the study and we will draw back to the question throughout this work.

A Brief History of Open Source

We need to reflect briefly on the various designations that the open source phenomenon has achieved. In this thesis, for purposes of simple English we will speak about open source as the generic domain of activity, abbreviated to OS, with OSS standing for open source software. However we must acknowledge that there are plenty of other designations in good currency, and they require us to present a little history. Some comprehensive historical accounts of open source software have been written (Moody 2001, Raymond 1999a, Rosenberg 2000, Weber 2004) and there is no intention here to attempt yet another.

The origins of open source software can be traced back to the hacker culture of the sixties when software was being sold together with hardware and macros and utilities were freely exchanged in certain user forums (Hars and Ou, 2000). In the early 1980s Richard Stallman, a researcher at MIT started to write a free UNIX system, GNU, and in 1984 founded the Free Software Foundation (FSF) (Ljungberg 2000). His work is recognised as providing the conceptual foundation for open source software as we know it today. Stallman who saw free software as having nothing to do with price but with rights (Stallman 1999a). In keeping with hacker tradition Stallman called his project GNU, a recursive acronym for 'GNU's Not UNIX'.

The free in free software is about freedoms and not about being free of charge – Stallman is oft quoted saying that "think of ``free" as in ``free speech," not as in ``free beer"" (Stallman, 1999). His definition of free software is about the ability of a user to have the freedom to "run, copy, distribute, study, change and improve the software" (Stallman, 1999). This definition incorporates four freedoms;

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbour.
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this[2]" (Stallman, 1999).

These freedoms were embodied in the copyleft method. Copyleft uses copyright law but "instead of using it as a means of privatizing software, it becomes a means of keeping software free" (Ljungberg, 2000). This creates a highly viral clause because the implication is that if a developer uses any part of GPL-ed code then the code s/he creates is also automatically GPL-ed as well[3].

It is clear that a key innovation concerning open source was Stallman's GNU General Public Licence [GPL] which made it possible "to legally improve and adopt software developed by others...[and] the open source licensing rules are important for learning and for creating derivative works" (Tuomi, 2002, p194). Indeed, as we will argue following Weber (2004), licences form the constitution for open source production. The GPL, for example, enshrines the rules of communication and openness allowed for any software under creation. Tuomi recognizes this but only goes as far as to explain that "an important role of copyrights ... is that they create norms that act as interfaces between developer communities" (2002, p201) stressing that this is how trust is crucially built up in such communities.

Stallman's Free Software Movement [FSM], and especially Stallman himself, are generally considered to be fanatical about free software and the GPL. But it was the word 'free' that forced a few people in 1997, including Eric Raymond, Bruce Perens, Larry Augustin, Sam Ockman to rename the movement (Perens, 1999). The free in FSM was considered to discourage businesses from taking up OS use and development[4]. The name 'open source' was coined by Chris Petersen and this change in the name was mainly the initiative of Todd Anderson, Chris Peterson, John "maddog" Hall and Larry Augustin, Sam Ockman and Eric Raymond[5]. This has led to two factions where Stallman leads the FSM and Raymond the Open Source Initiative [OSI]. Perens (1999) explains how Eric Raymond approached him at an invited conference, the *Hacker's*

[2] It is interesting to note how the basic freedoms are couched, especially in the terms of making changes to the source code. The source is an obligatory passage point for developers if they want to help develop ***it*** further.

[3] An interesting aside here raises the question that Feller and Fitzgerald named the Berkeley Conundrum (2002, pp173), "*if users do not actually download and modify the software source code, does it matter that it's open?*".

[4] Libre is the European word for free software and has been adopted by many to avoid any confusion over the meaning of 'free' (Anonymous 2005).

[5] http://www.opensource.org/docs/history.php

Conference, and then continued his conversation by correspondence in February of 1997. Raymond had come up with the idea of renaming free software to open source to make it more business friendly and steer clear of the taboo companies felt at anything to do with the word *free*. He began by approaching, what were then, small companies that dealt in Linux products and managed to convince Larry Augustin of VA Research and Sam Ockman. Thus began the *open source* movement and the Debian Free Software guidelines were adapted by Perens to become the open source definition and the name open source was registered as a trademark for the Open Source Initiative [OSI]. The OSI is controlled by Raymond and Perens (1997) and it was, and is, an 'organization exclusively for managing the Open Source campaign and its certification mark' (Perens 1999).

The 'open source' movement proper was thus initiated by people who were concerned that the word 'free' made their software unappealing to businesses. This can now be seen in its modified form on the Open Source Initiative website[6] where the definition has been divided into the main arguments it makes about the source code, derived works etc, and serves as the constitution of open source.

Focus of Study

The route chosen here to study the question of learning is through the ongoing debate among the members of the Linux kernel developers collective over whether to use version control software, and then which one to adopt. A version management system is software that "*manages and keeps track of the configuration items which are any documents created during a software development process, and which are found necessary to be placed under configuration control like requirements documents, data flow diagrams, design documents, source code, and test results*" (Kilpi 1997). This is true of version control software but it fails to emphasize its many other facets. In the current research the aim is to draw out the more interesting aspects of VCS and to reflect on how contentious and challenging a piece of software it truly is. It is only able to be so problematic because it has agency, material agency that it exercises to ensure its power and survival. Survival for all actors depends on evolving, adapting, or in a sense, learning.

A significant feature of version management system implicit in the above definition is that they act as an organizational memory and *prima facie* as a potential resource for learning, and as a basis for organizing. This debate of whether and which version control software to adopt within the collective is appropriate as a focus for this research because it has the ability to reveal how, when and why learning and organizing may occur, how learning and learning and organizing

[6] http://www.opensource.org/docs/definition.php

support is understood in the collective. Kogut and Metiu (2001) understand that open source activity exploits the intelligence in a distributed system and it is this 'intelligence' to learn, organize and evolve that this research addresses.

Theoretical and Methodological Orientation of Study

In this study a combination of Actor Network Theory, Bateson's concept of learning and Weick's concept of organizing are used to create a theoretical framework. Actor Network Theory [ANT] with its performative realist ontology sets the scene for studying the 'process of learning and organizing'. ANT is important to this study because it doesn't privilege human actors over material ones and thus provides a better lens to understand the role of version control software. This is aligned with Weick's similarly processual ontology of organizing with the added claim that an understanding [sense-making or learning] only comes to an actor in hindsight and after reflection. Bateson's simple yet powerfully flexible definition of learning as a change in behaviour is used because neither ANT nor Weick's ideas made explicit any exact definition.

The methodology of this study is strongly guided by ANT. ANT understands that technologies "indeed move through stages, have a chronology; that they may have setbacks that need to be overcome; that how they evolve is a function of background "macrosocial" factors of one kind or another as well as other relatively stable conditions in the real world; that there is more technological knowledge around at the end than at the beginning" (Law and Singleton, 2000). What is being studied gathers [and loses] certain features or factors on its journey, and the methods we employ to study also affect what is being studied. There are macro factors that are inscribed in the situation, technology etc but there are macro factors which affect the researcher as s/he gathers data. This is drawing back on Law's (2004) argument about how reality is created performatively but the methods engaged by researchers to understand this reality are also performative and affect the situation. Can actor network theory deal with both macro and micro issues is a question that has been asked before and often been claimed that there is too heavy a focus on the micro (Constant 1999) but as Law and Singleton (2000) explain in their reply to Constant all actors and especially technology are a product of the macro factors. To study them a researcher traces the network and this begins to unpack the macro concerns.

Seen from an ANT perspective the software development process needs to be transparent in order for the resources to lend themselves to recombination. The black box of the socio-technical artefact needs to be thrown open. In the open source process the openness is written into the software through the licence, for example the GPL (Lessig 1999a) and is implicit in the process too. Thus the source code is always available to view, modify and update. This makes resource recombination easier and maintains that the "open source model relies on mechanisms that effectively support recombinatorial innovation" (Tuomi 2002, p34).

Chapter Summaries

Chapter 1 provides insight into the motivation that provoked this particular study. The focus on open source and more pointedly on learning and organizing was stimulated by a curiosity in this phenomenon and how such widely dispersed actors are able to organize themselves and learn in order to survive. Chapter 2 maps out the main areas of research that have been studied to date in open source and places the current work in context. The main areas of research such as business models of open source, IPR issues, motivation of developers and structure questions are discussed before moving onto the four more specific areas related to this research. These four areas are technology in open source, license and governance of open source, learning and innovation in open source, and organizing and coordination in open source.

Chapter 3 introduces the first part of the theoretical framework, that of learning and organizing, using ideas of both Bateson and Weick. Bateson's ideas of learning and levels of learning are adopted which help to 'recognize learning when it happens' when studying the empirical data. Weick speaks of organizing as opposed to organization to stress the action and process aspect of his ontology. This helps to focus on movement rather than any stability because the claim is that there is no real stability as reality is constant change and movement.

This focus on process leads naturally into the next chapter, Chapter 4, which discusses the performative nature of learning and organizing using the terminology and methodology of ANT. The use of ANT is based on a premise that does not privilege human actors over and above material actors and thus allows a study of the social to emerge. Adopting the more recent approach taken by ANT (Latour 2005b, Latour and Weibel 2005), in particular by Latour, this study explains learning and organizing in open source as a socio-political unfolding over time and across space.

Chapter 5 explains the methodological route taken to carry out the empirical element of this research, linked tightly with ANT. The Linux kernel mailing list is studied to extract emails discussing version control software adoption. These messages are then coded using techniques of Grounded Theory but the attempt is not to create theory but merely to code the vast number of emails to allow the text to speak. This work is theory-laden and so only the techniques of Grounded Theory are used while keeping in mind Weick, Bateson, and especially the ordering narratives idea of ANT. The emphasis in this methodology is performative because it focuses on process, both the process of actor realities unfolding and the reality that the researcher feels emerging.

Chapter 6 details the case study of this research. The study of the adoption of version control software in the Linux kernel is told through the voice of the collective. We enter into the Linux kernel story in 1991 (though very little was discussed about version control software at that time) but the issue of adoption did substantiate in1995. Data was thus collected from 1995 to 2003 inclusive. The story of version control adoption leads the reader through the various challenges faced by the kernel collective of license struggles, exclusion from metadata, drive towards hybridicity through a gateway between different VCS, and finally the dubious monopoly (BitKeeper) VCS being snatched away from the collective, forcing them to create their own.

Chapter 7 and 8 together comprise the analysis of the data collected. While Chapter 7 discusses three micro themes of organizing time and space, sustaining the assemblage and machine agency, Chapter 8 draws all these together through the macro theme of transparency of learning and organizing. Of the three themes discussed in Chapter 7 organizing time and space through version control software conveys the message that VCS manages time and space and so allows the collective to revisit time, it has the ability to manage time is necessary and displace control in the collective, and that its capacity to fold time is critically dependent on its ability to hold and manipulate code.

The second theme, that of sustaining the assemblage, relates how code creates a collective around itself, keeps the collective together and sustains it; the future is hybridization, and not either extreme of closed or open source because code is practical and there is little room for fanaticism; and finally that technology, in this case VCS, manages the rather delicate balance between learning and organizing in open source. Theme three in Chapter 7 is machine agency. It is an idea which is key to this study because it develops the concept of the fluid nature of agency and how it dances between actors. VCS is structured as space in a way that increases the ability of the macro actor to control the collective, and this is somewhat mitigated through collective ownership of the software because at the same time it provides transparency, and can be a vehicle for collective learning.

Chapter 8 folds the three themes mentioned above into the larger encompassing theme of transparent learning and organizing. This chapter explains how efficient organizing and effective collective learning require transparency and openness of process and structure. VCS thus behaves rather like a marketplace. It then goes on to explain how transparency, through the use of VCS, makes apparent '*becoming in action*'. VCS use in the Linux collective creates a balance between learning and organizing, which can be understood as '*evolving organizing*' or learning *through* organizing.

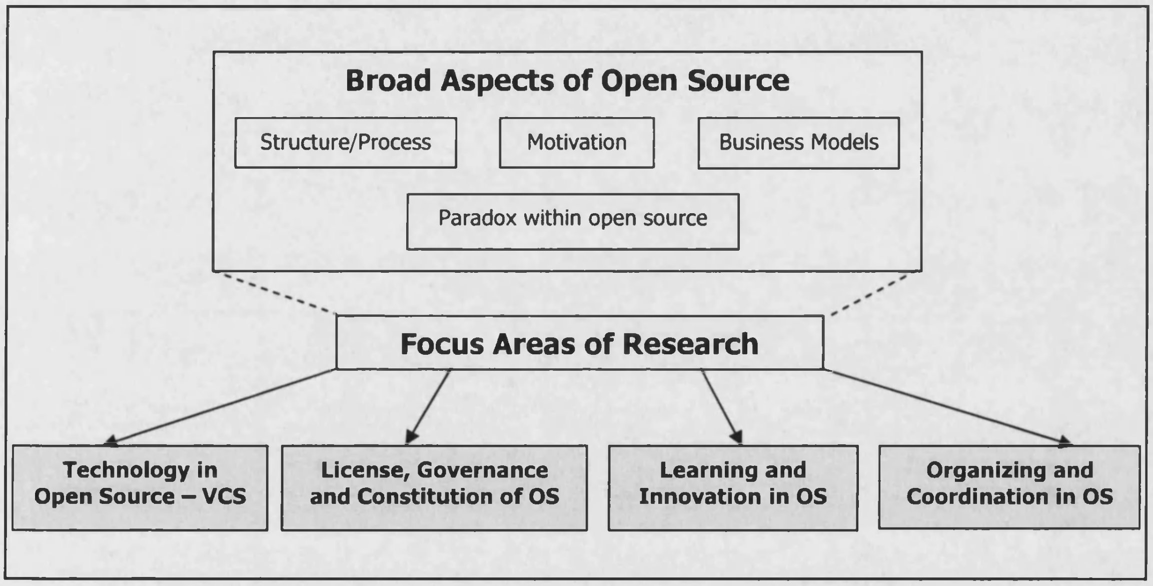
Chapter 9 returns attention to the research question of this work and methodically frames an answer based on the two analysis chapters. It steps through how learning occurs and is manifested in open source collectives and then follows these sections with a focus on organizing and manifestation of organizing in open source collectives. It then draws organizing and learning together in open source to show how it is only through a combination of the two concepts that we can better understand how either occurs in such collectives. The link between the two, this research asserts, is in the explanation and transparent unfolding of *becoming* which sways the collective towards organizing at times and at other times towards learning. However, the primary carrier of this process, we stress is code. This brings us back a full circle from studying code to understand learning and organizing to find that is the code that learns along with other actors and that it organizes other actors in the collective around itself.

Chapter 9 also details the main contribution of this research. The theoretical contribution of operationalizing the constructive realist ontology of ANT is coupled with adapting Bateson's ideas of learning to stretch to material actors as well. The methodological contribution made is an explanation of the rigorous analysis of a large amount of archival data through ANT tools and textual analysis. And finally, the practical contribution for open source and other such collectives can be considered as the varied, and we claim, better understanding of the role version control software plays in a collective. It is not a simple tool but a far more contentious, power and control seeking actor which knows that it must continually evolve to sustain itself and its position in the collective.

Chapter 2
The Study of Open Source

Introduction

This chapter presents a literature review of research in the field of open source. Figure 1 outlines the structure of the chapter. First we address the broader accounts of open source before we focus on the specific areas that provide the necessary background for the current work, the technology used in open source development; the effect of the license on the governance structure of a collective; learning and innovation research in open source (developed further in Chapter 3); and studies of organizing and coordination.



Broad Aspects of Open Source
Structure/Process
Motivation
Business Models
Paradox within open source
Focus Areas of Research
Technology in Open Source – VCS
License, Governance and Constitution of OS
Learning and Innovation in OS
Organizing and Coordination in OS

Figure 1: The Structure and Focus Areas of Open Source for this Research

Most early literature on open source incarnated the ideology of 'hackers' as central to open source and, as most of the famous work written was by free/open source programmers and leaders themselves, like Raymond, Stallman and O'Reilly, this forces one to question the objectivity of the work (O'Reilly 1997, O'Reilly 1999a, O'Reilly 1999b, O'Reilly 1999c, Raymond 1998a, Raymond 1999a, Raymond 1999b, Raymond and Trader 1999, Stallman 1984, Stallman 1999b). The desire here has been to steer clear of the regurgitated ideological fare presented by most of the 'hackers' themselves. Indeed the intention is to go beyond the rhetorical accounts of open source as a better way or a model for organizing and subject it to a more detailed and probing study.

Since the early period a body of more independent research has emerged, but still most OS work can be seen, implicitly or explicitly, to reflect a sense of challenge, a new departure or a radical turn. This fundamental characteristic leads to much writing expressing the phenomenon in terms of a clash of interests or a battle – for example, in the many detailed accounts of Microsoft vs Linux (Garcia 2001, Moody 2001, Valloppillil *et al.* 1998a, Valloppillil *et al.* 1998b). At other times this sense of a radical turn is used as a metaphor or exemplar for a wider process of change in how organisations or markets might work, how organising may be achieved in both societal and organisational terms, and how interests of producer and consumer are redefined in a digital economy (Ciborra 2000, Markus *et al.* 2000). In such work it is generally understood that the technical and social aspects are intertwined and so must be dealt with side by side. Still, as an endeavour essentially bound up with technology and a process of technological production, the actual means used (process) is of considerable interest.

In most literature the collaboration-based process of OS is usually seen to stem from the academic background of the early hackers, including Richard Stallman of MIT and Linus Torvalds at the University of Helsinki. Thus OS is seen to share many features with academic peer-reviewed work. Eric Raymond presented the most influential early analysis of the OSS process in his *Cathedral and the Bazaar* (1998b). Here he, rather romantically, likened the OS process to a bazaar where a cacophony of ideas is exchanged. This famous analogy laid out some guiding principles of the 'bazaar model' (essentially a process model not an economic model) concerning how a project begins (by scratching an itch), its reliance on reusability, parallel processing, throw-away prototype modelling, and the choice of a successor.

Structure and Process of Open Source

Much literature sees open source collectives as a cohesive community (Butler et al. 2002, Gallivan 2001, Himanen 2001, Sharma et al. 2002), while others are more interested in how such developers create quality software (Kesan and Shah 2002), and what is novel about the software process (Bowman et al. 1999, Crowston and Scozzi 2002, Jorgensen 2001, Raymond 1999a). Bergquist and Ljungberg (2000), for example, describe OS as a form of virtual community on the Internet. They suggest that the community has developed an advanced culture for 'information sharing, online co-operation and organizational learning that is structured without formal bodies but with the help of a very strong clan-like "gift culture"' (Bergquist and Ljungberg 2000, Bergquist and Ljungberg 2001). Though a rather rosy description of OS, it is missing the underlying passion of OS work which Himanen (2001), captures well with his 'passionate and freely rhythmed work'. It is this passionate phenomenon with mysterious motivations and rewards that has inspired much recent research, a quality caught well by Raymond (1998b) "No quiet, reverent cathedral-building here – rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches.... out of which a coherent and stable system seemingly emerges only by a succession of miracles".

The 'community' concept has been debated by Ciborra (1996) and Markus et al (2000), who suggest rather that OS should be seen as a new organizational form. Yet the word community has become the most persistent way of describing this phenomenon. Tuomi (2000, 2002), drawing on Fleck's work, uses a combination of the terms organization and community to explain how OS, and in particular the Linux developers, are a 'fractal' organization. Tuomi (2002) describes a fractal organization as a "community of communities...where people are members of a broad 'Linux' community at the same time joining one or more of its sub-communities. These communities are organized around central 'gurus', 'old-timers', and more peripheral novices who have been accepted as legitimate members of the community". Drawing on Fleck (1979) Tuomi also used the concept of a collective – a thought collective – defined as a "community of persons mutually exchanging ideas or maintaining intellectual interaction".

As explained in Chapter 1, we understand the Linux kernel actively as a collective which "unlike the words 'group' or 'organization' refers to individuals who act as if they are a group. People who act as if they are a group interrelate their actions with more or less care, and focusing on the way this interrelating is done reveals collective mental processes that differ in their degree of development" (Weick and Roberts 1993, p360).

Motivation

One of the earliest areas of academic study relating to the structure and process of OS explored participants motivation (Hars and Ou 2000). Their more elaborate study (Hars and Ou, 2001; 2002) became the basis for many studies that followed (Lakhani *et al.* 2003), Lakhani and Wolf (Lakhani and Wolf 2005), Hertel et al (2003) and Rossi and Bonaccorsi (Rossi and Bonaccorsi 2005b). Hars and Ou (2002) used Deci's (1975) classification between intrinsic and extrinsic motivation where intrinsic concerns internal psychological factors and extrinsic is about external rewards. They make a comparison between Deci's and Maslow's (1959) ideas of motivation but preferred the former for their study, and most researchers have followed suit.

	Motivations	Sources
MICRO	Open Source Individual Level Motivation	
Extrinsic	Low opportunity costs	Bonaccorsi and Rossi (2003); Kollock (1999a); Lakhani and Hippel (2003)
	Monetary rewards	Feller and Fitzgerald (2002); Hertel, *et al.* (2003); Lerner and Tirole (2002b)
	Reputation among peers	Dalle and David (2005); Lerner and Tirole (2001); Bezroukov (1999)
	Future career benefits	Lerner and Tirole (2004); Fielding et al (2002); Lee et al (2003); Ljungberg (2000)
	Learning	von Krogh, *et al.* (2003b); Dutta and Prasad (2004); Edwards (2001); Lakhani and von Hippel (2003)
	Contributions from the community	Raymond (2001); Bonaccorsi and Rossi (2003); Roberts et al (2004).
	Technological concerns	Weber (2004); David and Pfaff (1998) (1998); Pavlicek (2000)
	Filling an unfilled market	Feller and Fitzgerald (2002); Green (1999);

(2003)
Bitzer, *et al.*, (2005); Zeitlyn (2003)
Crowston and Howison (2004); Hars and Ou (2002)
Stallman (1984)
n Level Motivation
Lerner and Tirole (Lerner and Tirole 2002a)
Feller and Fitzgerald (2002); Wichmann (2002b); Lerner (2002)
Lerner and Tirole (2002b); Wichmann (2002b)
Hawkins (2002); Lakhani, *et al.* (2003); Dahlander (2004)
Aoki (2001); von Hippel (2002); Fink (2003)
Fink (2003); Wichmann (2002c); Lerner (2002); Henkel (2004)
Feller and Fitzgerald (2002); Tuma (2005)
Fink (2003)
Kuster, *et al.* (2002); Lerner and Tirole (2002a)
Kuster, *et al.*, (2002); Franck and Jungwirth (2002)
Feller and Fitzgerald (2002)
Feller and Fitzgerald (2002)

lapted from Bonaccorsi and Rossi (2004))

	Altruism
	Sense of belonging to the community
	Fight against proprietary software
MACRO	Open Source Fir
Extrinsic	Independence from price and license policies of large software companies
	Supplying software–related services
	Indirect revenue from selling related products
	Exploiting R&D activity of other developers and OS firms
	Software testing by users' community
	Hiring good Open Source technicians
	Lowering hardware costs
	Security concerns
Intrinsic	Conforming to the values of the OS community (not betraying the trust of other developers)
	Code sharing with the community (reciprocating to sustain cooperation)
	Fight for software freedom (reducing the market shares of large software companies)
	Affordable software for developing countries (reducing digital divide)

Table 1: Macro/Micro Motivation in Open Source (A

ho claim that "signaling – … rarely features

The distinction is also made by Bitzer *et al.* (2005) w intrinsic motives that have been discussed in the wider OSS literature: (a) user programmers that actually need a particular software solution, (b) the fun of play or mastering the challenge of a given software problem, i.e. homo ludens payoff, and (c) the desire of belonging to the gift society of active OSS programmers" (p12). Nonetheless, other motivation studies have shown how developers work to gain a reputation (signalling) that could then lead to better job prospects and real financial gain. Head-hunters for software development companies, for example, regularly scan the credits list of OS products in order to discover talent, which could then lead to offers of good jobs and pay (Hars and Ou 2001, Hertel *et al.* 2003). Others emphasise the concept of a gift-culture, though this is not without its demands of reciprocity. The giving of a gift may seem like an altruistic gesture yet the implication behind gifts is that one day the receiver will have to respond likewise, not to mention the underlying power which comes with the ability to make gifts (Bergquist and Ljungberg 2001, Mauss 1954).

Feller and Fitzgerald (2002) make their own macro/micro distinction when discussing motivation. The micro involves individual level motivation and the macro concerns the collective incentive. Three broad categories of motivation are proposed within the macro/micro division; technological, social-political, and economic. Rossi and Bonaccorsi (2005a, 2005b) interpret the macro issues to indicate firm or organization level motivation for open source software adoption or development and provide a comprehensive overview of literature in open source motivation [see Table 1].

Business Models and Economic Aspects of OS

Another strand of research that links to structure and process is that of business models. There is literature on OS in relation to particular markets or environments (Mockus *et al.* 2002) and the role of media and infrastructure (Potter 2000). Academic studies here try to make sense of how open source manages to exist as a viable business model (Johnson 2001, Lerner and Tirole 2000, Ljungberg 2000, Weber 2000, Weber 2005b). Some influential business models include Ghosh's (1998) 'cooking-pot' model. It explains that people put into the pot what they have, the cooking pot combining and cloning them, and people then taking out what they need. This cooking pot model is viable if people see that they get more out than they put in, and he notes in particular that what they take out and value is the diversity of things to be found in the cooking pot. If these conditions generally hold, the cloning of inputs, the valuing found in diverse outputs, then problems of free riders are not catastrophic, or put more helpfully, positive network externalities are seen. As in other accounts of OS, Ghosh sees reputation as the primary form of value that drives this process; and quotes Linus Torvalds in support, 'Yes, you can trade in your reputation for money," said Torvalds, "[so] I don't exactly expect to go hungry if I decide to leave the

University. 'Resume: Linux' looks pretty good in many places". (Subsequently, we know Torvalds did leave the University and moved to the USA – see Moody (2001) for this and other OS pioneers who have traded their reputation.)

This cooking-pot model may (misguidedly Ghosh suggests) lead us to believe that the trade of ideas on the Internet smacks of altruism, but he notes, this would be missing the point. He insists that a participant in an OS community is not offering an idea for free, but knows that there will be recompense; the only uncertainty is the exact time of claiming a return. If we have other people doing the same it keeps the pot 'cooking', as some are taking and others are adding. Still, everybody knows that if they stop contributing then there will be a time when the pot will run dry (indeed the fate of most OS projects).

Another model, offered by Benkler (2002), he names commons-based peer production. He suggest that peer-production will thrive where three characteristics are present: *modularity* to allow parallel work, *finegrained work activities* to allow participation of many with a little to give (a little motivation), and finally, integration of fine grained modules must be efficient, otherwise the integrator will appropriate the residual value (p8). The 'central characteristic is that groups of individuals successfully collaborate on large-scale projects following a diverse cluster of motivational drives and social signals, rather than either market prices or managerial commands' (Benkler 2002). This places the commons-based peer production model midway between either extreme and appears to provide a better explanation of open source production and continued existence.

Tuomi (2002, p28) uses de Certeau's (1988) work on the concept of *la perruque* to explore the relations of OS to the business world. *La perruque* translated literally means 'the wig' and alludes to the idea of a disguise. *La perruque* is about using resources of your place of employment for work that is not for your employer. De Certeau differentiates this from absenteeism or stealing from the employer because *la perruque* is about using the resource, time mostly, to work on something that is not for the employer but during work hours. Many open source developers are "guilty" of this but some are fortunate enough to have this allowed and written into their employment contract. If *la perruque* is the reason for open source developers being able to support themselves while they work on what they enjoy then this could explain, at least economically how it is viable for them to work 'for free'. Tuomi pushes this concept further to explain how open source developers exploit the resource of time at work because there is usually a certain amount of slack or space built into organizations. A certain amount of slack is encouraged or at least ignored because it is believed that this can lead to innovative activities that will eventually benefit the company, or to entice employees to work for you. Tuomi (2002, p28) argues that “a ‘perfectly efficient’ organization has full accountability and predictability and no slack, and therefore no space for new interpretations or practices”.

With increased business interest in open source a more recent research focus has been directed to company level adoption of open source (Bonaccorsi *et al.* 2004, Grand *et al.* 2004, Wichmann 2002a). Grand *et al.* (2004) create a multi-level management model of increasing private resource allocation through, paradoxically, greater public investment. Their ‘ladder’ model explains how and why companies engage deeper and deeper into open source software production and business models where “each level implies greater outlay of private resources and increased dependency upon publicly available knowledge assets” (p591). The four steps of the ladder are; firm as a user of OSS; OSS as a complementary asset; OSS as a design choice; and finally the highest level of the ladder is when a firm adopts an OS compatible business model [see Figure 2].

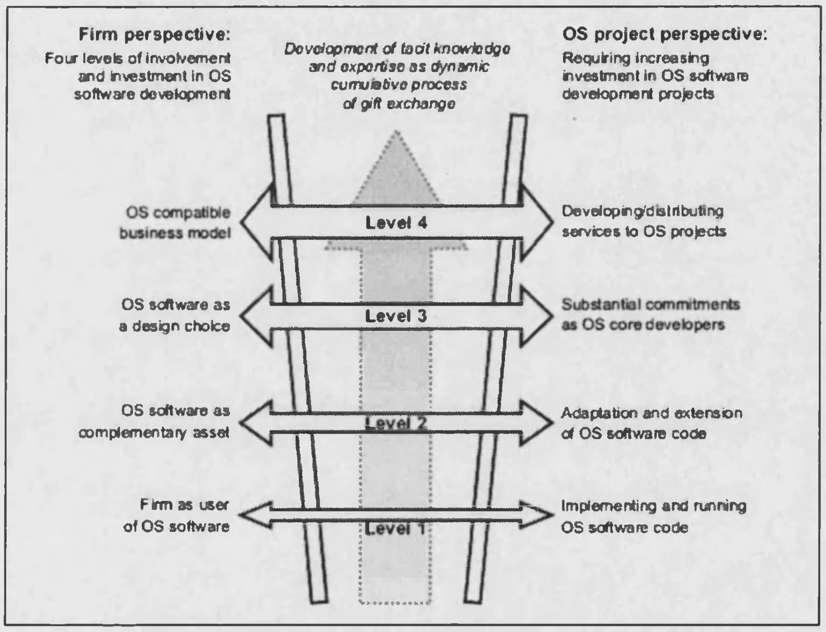

Firm perspective:
Four levels of involvement and investment in OS software development
Development of tacit knowledge and expertise as dynamic cumulative process of gift exchange
OS project perspective:
Requiring increasing investment in OS software development projects
OS compatible business model
Level 4
Developing/distributing services to OS projects
OS software as a design choice
Level 3
Substantial commitments as OS core developers
OS software as complementary asset
Level 2
Adaptation and extension of OS software code
Firm as user of OS software
Level 1
Implementing and running OS software code

Figure 2: The Ladder: A four-level management model of resource allocation (Grand et al, 2004, p595).

Paradox of OS

Fitzgerald and Feller (2001) note in their introductory editorial to a special issue of the *Information Systems Journal* that “opinion on OSS tends to be quite polemic”. Open source manages to court controversy but this is probably because it lays bare many questions, questions and confusions which are compounded by hacker indecision over the reason of their apparent success! Wayner (2000, p118), mentions three problems [which are related to each other] with the OS model; making collective decisions; limited desire to work on less attractive or boring but needed projects; and maintenance (documentation). He uses an example of changing a lightbulb to describe the first, ""How many open source developers does it take to change a lightbulb?" The answer is: 17. Seventeen to argue about the license; 17 to argue about the brain-deadedness of the lightbulb architecture; 17 to argue about a new model that encompasses all models of illumination and makes it simple to replace candles, campfires, pilot lights, and skylights with the same easy-to-extend mechanism; 17 to speculate about the secretive industrial conspiracy that ensures that lightbulbs will burn out frequently; 1 to finally change the bulb; and 16 who decide that this solution is good enough for the time being". This illustrates how hard it can be to convince OS developers to decide on the best approach and get to work. Collective decision making processes often suffer from such futility but is there something distinctive and unique about OS?

Other well-cited issues with OS development include the problem that OS developers are more than keen and capable of producing innovative and exciting software, yet can hardly ever be made to work on a task that is dull, repetitive or not sexy; "fine-tuning a graphical interface, or making sure that the scheduling software used by executives is as bulletproof as possible" (Wayner 2000, p118). This draws into the next problem highlighted by Wayner (2000), that of maintenance and the lack of documentation which causes developers and users real anxiety. While this is a problem that besets software development in general it does pose some distinct challenges for open source, (e.g. given the primacy of code).

Maintenance is a slightly different issue in open source if (ideally) the users are the developers and the maintainers too. Indeed it may be easier to explain maintenance as straightforward 'more development' if open source projects are kept alive by constant improvement and work. This can be viewed as lifelong maintenance or just development. Clearly the four classic types of maintenance, adaptive, corrective, perfective and preventive (Swanson 1976) all tie into the activities of open source and are a part of the open source development process. But as Rodriguez *et al* (2004) point out, "maintainers must know what changes [they] should do to the software, where to do those changes and how those changes can affect other modules of the system. Frequently they do not have enough knowledge to make the best decision and must consult other information sources, but these sources are often unknown or difficult to locate". This is where they believe that version control software plays an important role.

Fitzgerald and Agerfalk (2005) see OS as innately paradoxical as a result of its openness of source code, and explore six paradoxical issues that result; the extent to which the cathedral and bazaar metaphor holds true for open source; the collectivist versus individualist question of motivation and contribution of open source developers; the apparent tensions and in-house quarrelling between various factions like OSS and FSF; the extent to which open source truly reflects a departure from software engineering principles of production and distribution; and the question of so-called better quality software creation in open source. They examine the trend and analyze whether open source has the potential to apply to areas other than software development. They also question whether a name change from free to open source software will prove to be enough to make such software development attractive to businesses. The latter issue especially leads to the concept of what Fitzgerald (2006) calls OSS 2.0. Fitzgerald explains that there is a strong trend and evolution of FOSS towards a more commercially aware and viable hybrid. Indeed, the term open source was introduced in 1997 to create the potential for such business viability, but Fitzgerald provides a finely teased out distinction between FOSS and OSS 2.0 to explain the subtle transition. However, what is less obvious in this work is any comparison between OS and OSS 2.0.

Technology in Open Source

Relatively speaking, the literature on open source is less well developed in considering the technology and tools used. Often they are seen as being simple and obvious choices made on the basis of needed functionality and posing few substantial issues. Still, open source developers not only create software technology but their process of development is heavily reliant on it for infrastructure, tools and communication e.g. Emacs, GVim, Netbeans, GNU Make etc (Bittman *et al.* 2001). Some face-to-face interaction does take place but the largest part of communication is carried out via mailing lists, websites, email and chat group sessions. But, for example, mailing lists perform more than just a facilitation of communication role; when they are archived (e.g. the Linux kernel LKML studied here) they form a collective or organizational memory and can be "searched when someone needs to know whether something is known about a potential bug. In that way the mailing list archives provide a simple but effective form of community memory" (Tuomi, 2002, p193).

In this section we consider in particular version control software (VCS) (Berliner 1990, Fogel 1999, Rochkind 1975, Tichy 1985, van der Hoek 2000), one of the most ubiquitous technical elements in wider OS activity, though, as we shall see in Chapter 6, problematic for Linux. Thus Feller and Fitzgerald (2002, p95) write that VCS [CVS] is the "most mission-critical OSS tool".

Version control software is part of the basic eyeballing of bugs and patches that is encouraged in open source and has been proposed as a 'law' by Raymond – Linus's Law – "Given enough eyeballs, all bugs are shallow" (Raymond, 1999). The wider potential and role for VCS is well captured by Louridas (2006, p104) as allowing developers to "retrace and restore our programming past, proceed along two or more development lines in a single project, and coordinate our work with the work of others in the same project".

Authors such as Spinellis (2005) and Atkins et al (2002) are emblematic of the tool like lens through which version control software is often understood – simply an unproblematic tool and no more. Robbins (2005) similarly describes various 'tools' used by open source collectives but again we have the very apolitical discussion about technology. He does however, provide a useful categorization of tools consisting broadly of seven areas; version control (e.g. CVS, Subversion); bug tracking and technical support (e.g. bugzilla, Scarab); technical discussions and rationale (e.g. Mailing lists, project websites); build systems (e.g. Gump, Automake); design and code generation (e.g. ArgoUML, Torque); quality assurance (e.g. Flawfinder); and collaborative development environments (e.g. Sourceforge). Louridas (2006) comes closest to approaching the political dimensions of technology through his discussion of the issue of vendor lock-in and how this is a particular concern when adopting version control for a project. He stresses the importance of version history and how vendor lock-in can leave you bereft of your past work if the repository is not in a portable format.

Tuomi (2002, p23) goes on to suggest that the drivers for innovation are "often found by looking for tensions and contradictions in existing social practice" and that "technology addresses a need when it releases or reduces some of the tensions generated in this process". However, this ignores that technology is often the reason and source of contradiction and often innovation too. We develop this idea further in this research with a focus on the process of version control software adoption.

Version Control Software

Software configuration management has been defined as the discipline of controlling the evolution of complex software systems. "Configuration management is the controlled way of leading and managing the development of and changes to combined systems and products during their entire life cycle" (Asklund and Bendix 2002). Configuration management is used by both proprietary software developers and open source but is perhaps more central to the OS process (van der Hoek 2000).

Version control is a mechanism by which multiple versions of any software can be managed, kept track of and protected against overwriting (Clemm 1989, Kilpi 1997). A schematic overview of such systems is given by Keats (2003). Such a tool in many cases has provision for adding metadata (data about code) and acts as a repository of code. It is usually referred to as a tool but here we describe it as *software.* This is a more appropriate term because our ANT stance seeks to give it equal status as an actor. The term *tool* relegates it to a simple artefact which has no agency or control.

UNIX has had a history of using such tools from the start, the most popular have been RCS [Revision Control System] or SCCS [Source Code Control System] (Koike and Chu 1997a, Koike and Chu 1997b) but CVS [Concurrent Versions System] is the tool which has gained the most popularity in OS and it would now be hard to find an OS project which doesn't draw on CVS or one of its derivatives (van der Hoek 2000). The UNIX community needed a tool to "*manage software revision and release control in a multi-developer, multi-directory, multi-group environment*" (Berliner 1990). Add to this distributed development and one can see just how integral coordination mechanisms are in geographically distributed development over and above collocated development (Herbsleb and Grinter 1999).

The earliest form of version control software was the *diff* and *patch* programs. The diff program on its own provided the changes that had been made to a file, or the difference, thus the name diff. Larry Wall then created the *patch* tool to ameliorate this problem and his work was later added to by Paul Eggert (MacKenzie *et al.* 2002). Developers now had both the diff and the original patch which enabled them to compare two or more patches line by line for changes. In order to be able to see the entire code a developer could add the diff to the patch and realize the most recent improvements. The diff and patch programs were followed by Marc Rochkind's *Source Code Control System* (SCCS) in 1972 developed at Bell Telephone Laboratories (Bolinger and Bronson 1995). SCCS was an improvement on the diff and patch method because it converted all changes into deltas which could then be strung up together to form the complete software (Rochkind, 1975). The diff and patch tool didn't provide the flexibility of being able to access various changes, one could compare two pieces of code but each delta (patch) was not a separate file with a date stamp and other information.

Walter Tichy's *Revision Control System* (RCS) in the early 1980's was yet a further enhancement in version control software. It allowed for both forward *and* reverse deltas (both adding and removing changes). Its 'lock-modify-unlock' method authorized a developer to lock the tool thus disabling any other developer from making simultaneous changes, to make his changes, and then unlock the tool for others (Fogel, 1999). The locking ensured that no patch was overwritten by two or more developers adding their work at the same time. Improvements offered by RCS in the form of reverse deltas, branching in versions, and 'stamping' of revisions with unique markers for easy identification of particular version (Tichy, 1991) were outweighed somewhat by the constraint of locking other developers out. Dick Grune of Vrije Universiteit in Amsterdam created patches to add to RCS which became the first form of *Concurrent Versions System* (CVS), "The initial commit of CVS under itself was on 1985/11/23 23:24:37. I posted the improved scripts to comp.sources.unix (and so implicitly to mod.sources) on June 23, 1986, by sending them to one "Rich" at sources@mirror.UUCP. Two years later Brian Berliner picked it up, turned it into C code, and the rest is history" (Grune, 2003).

Brian Berliner's enhancements of CVS were created under the GPL so that the community could benefit from this tool (Berliner, 1990). CVS, according to Berliner was a "program…[that] fills a need in the UNIX community for a freely available tool to manage software revision and release control in a multi-developer, multi-directory, multi-group environment. This tool also addresses the increasing need for tracking third-party vendor source distributions while trying to maintain local modifications to earlier releases" (Berliner, 1990). Using an innovative copy-modify-merge model, CVS eradicated the need to lock developers out of the system. Instead, developers would make a personal copy of the program or file, modify and work on that, until they were ready to merge their changes with the master repository. This allowed for simultaneous work by

[redacted]

would try and merge their respective change on the same file. When this
flag the 'conflict' and ask the second developer to resolve the issue by choosing
uld be allowed. CVS was made "network-aware" in the 1990's by Jim Kingdon,
rs gained "networked access to sources, simultaneous development, and
ted merging of changes" (Fogel, 1999).

a number of other version control programs written between CVS and the
ding Arch, Subversion, ClearCase, PRCS[7] but for the Linux kernel community
it has made an impact. *BitKeeper* is owned by the company BitMover and was
Larry McVoy[8] in 1999, in an attempt to create software that would suit the
rvalds and the Linux kernel in particular. What sets BitKeeper [BK] apart from
version control programs mentioned above is that BK is a proprietary tool and
available along with the product (for more detail on BK see Chapter 6).

er of advantages over CVS and other tools. It groups changes into changesets
he developer to make single changes, and these changesets enable 'automatic
o that developers have an easy method of backtracking to specific changes
to trace each single change and branching of the program (Henson and Garzik,

happens CVS can
which change sho
so that develope
intelligent automa

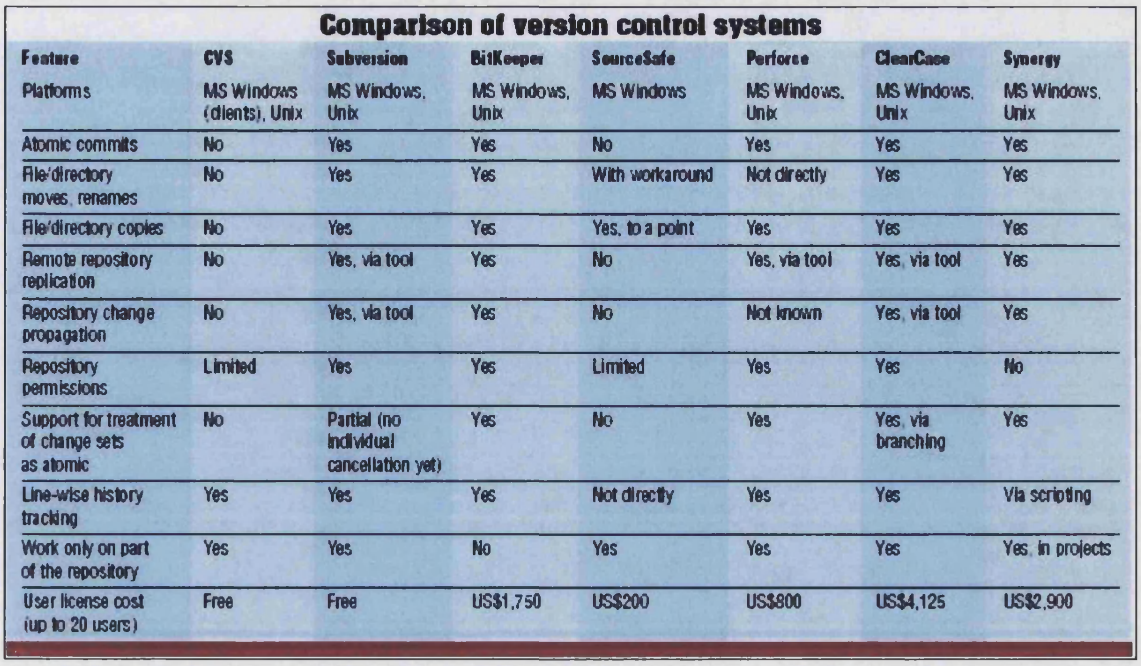
There have been
present time, inclu
it is BitKeeper tha
created mostly by
needs of Linus To
most of the other
source code is not

BK claims a numl
rather than force t
synchronization' s
rather than having

2002). It also, somewhat like CVS, allows for a copy of the master program to be made, called a clone in BK, and it has a push-pull method of control,. However the key attraction of BK is its auto-merging algorithm, though there is still the possibility to perform manual merges as well. A developer can perform two-way as well as three-way merges[9] with BK. This tool also allows for metadata to be held along with the patches of code. Metadata "is information about the data managed by the BitKeeper Software in a BitKeeper Package, such as the ChangeSet file; the messages which annotate modifications of the data (also known as check in comments, ChangeLog entries, and/or log messages); and all files contained in the top level BitKeeper directory in a BitKeeper Package, in particular the BitKeeper/html directory and the BitKeeper/etc/config file" (Corbet 2002, BitKeeper License version 1.38, created on 28/03/02)[10].

Henson and Garzik (2002) (both Linux developers, Val Henson one of the few female Linux hackers) claim that BitKeeper has a superior architecture to traditional source control systems and it is this feature which attracts Linux developers. Developers want to be able to commit work

[redacted]



Comparison of version control systems

Feature	CVS	Subversion	BitKeeper	SourceSafe	Perforce	ClearCase	Synergy
Platforms	MS Windows (clients), Unix	MS Windows, Unix	MS Windows, Unix	MS Windows	MS Windows, Unix	MS Windows, Unix	MS Windows, Unix
Atomic commits	No	Yes	Yes	No	Yes	Yes	Yes
File/directory moves, renames	No	Yes	Yes	With workaround	Not directly	Yes	Yes
File/directory copies	No	Yes	Yes	Yes, to a point	Yes	Yes	Yes
Remote repository replication	No	Yes, via tool	Yes	No	Yes, via tool	Yes, via tool	Yes
Repository change propagation	No	Yes, via tool	Yes	No	Not known	Yes, via tool	Yes
Repository permissions	Limited	Yes	Yes	Limited	Yes	Yes	No
Support for treatment of change sets as atomic	No	Partial (no individual cancellation yet)	Yes	No	Yes	Yes, via branching	Yes
Line-wise history tracking	Yes	Yes	Yes	Not directly	Yes	Yes	Via scripting
Work only on part of the repository	Yes	Yes	No	Yes	Yes	Yes	Yes, in projects
User license cost (up to 20 users)	Free	Free	US$1,750	US$200	US$800	US$4,125	US$2,900

Figure 3: Source - Louridas, 2006

[redacted]

Tuomi does relate learning with creativity (innovation), "Quality control in innovative and continuously evolving projects is essentially about learning. Whereas the traditional models of learning in product development focused on decreasing errors in a given product design, in the case of Linux learning is also creative" (Tuomi, 2002, p193). However, he uses terms like learning, innovation, creativity etc interchangeably and rather loosely. In the next chapter we focus in depth on learning and organizing. In the rest of this section we briefly review two useful contributions that directly relate to learning in open source (von Krogh *et al.* 2003a) and its distribution (Weber 2005a).

Some have explained learning as essentially a part of the communal resource (von Krogh *et al.* 2003a) which is created through "the production process of knowledge in an open source software project [and] has as a byproduct communal resources that reward its contributors"(p10). Von Krogh *et al.* seem to believe that learning opportunities are simply a by-product of the development process. Of the three rewards they discuss the last (learning opportunities) is our focus but the explanation of the second (control technology) given by the authors merits mention, if only as an opposing view to the one we stand by in this research recognizing non-human agency. When describing learning opportunities they manage to draw an intriguing connection between learning and level of control over change and decision-making. They 'manage' because this link is not consciously drawn out, but it seems an obvious conclusion from their data analysis that could have been usefully expanded.

We now turn our attention to the distribution of learning. This is an issue in this study because VCS not only facilitates the creation of knowledge but plays a pivotal role in distributing it in the collective. Weber (Weber 2005a) attempts to create a loose framework which incorporates learning creation with distribution. His ideas include; "design for evolution (allow the community to change); open a dialogue between inside and outside perspectives (tightly insulated communities tend to corrode); allow for different and bursty levels of participation (different people will participate at different levels, and any single person will participate at different levels over time); preserve both public and private community spaces (not all community interactions are public; backchannels should be available); focus on the value that is created for the people in the community; mix the familiar and the new; and facilitate the creation of a rhythm (pure bursty-ness and unpredictability tend to corrode commitments)". These ideas are interesting because they help to place VCS in a broader context but a VCS may qualify directly on a number of these needs mentioned.

Organizing and Coordinating in Open Source

Ljungberg's (2000) seminal paper on organizing, though not the earliest, maps out a comprehensive understanding of organizing in OS. Ljungberg raises a number of questions for further research and we have taken up some of the challenge in this thesis. The most intriguing point, which is often cited by others but never truly thought through, is how open source may be key to understanding the future of organizations, and how they organize. For example, he explains the Linux kernel as a 'mixture between hierarchy and market… [but it is] loosely coupled in the form of a community regulated by norms rather than rules' (p214).

There are three distinct accounts of organizing presented here – as gift and people, as market and environment, and a coordination and modularity. Each one furthers our understanding of how organizing happens in open source collectives but from our perspective misses a key issue, that of the role and agency of material actors.

Organizing has been explained in the OS literature in terms of gift-giving as a means that 'organizes relationships between people' (Bergquist and Ljungberg 2001). Gift-giving is not seen as altruistic; rather it provides the giver with the ability to attract attention and move vertically in the hierarchy of collective power. The authors add that the giver-user dynamic is a process which changes and evolves over time where at one point in time a particular actor through his/her contribution has a more influential role. This work raises some very interesting claims but the insistence on only human actors being capable of playing games to harness greater influence disappoints and thus we stress a need to study the social in totality.

Weber (2004, p37), drawing on Sabel's (1982) words, explains how it is possible that both proprietary and non-proprietary forms of software development are able to coexist because they are "different ways of organizing work, "based on different markets, rooted in correspondingly different patterns of property rights," [and thus they can] simultaneously prosper. Neither is a technological necessity, and neither can claim to have "won out" in any meaningful sense". What interests us in the current work is how proprietary and non-proprietary forms co-organize, which is distinct from them co-existing. In our case we have the BitKeeper License and GPL both trying to out-organize each other in an attempt to take control over the collective. Koch and Schneider (2002) allude to the idea that open source projects, like proprietary ones, will need to adopt a version control system if the project grows fairly large. Iannacci (2003), using the example of BitKeeper use, a particular version control system in Linux describes how the managing model 'leans towards adaptability rather than adaptation'. The latter creates a sense of ongoing change and links in well with our performative perspective on learning and organizing.

Crowston and Howison (2006) studied three different repositories for bug fixing interactions and found that in Sourceforge, GNU Savannah and Apache Bugzilla the structure became more modular and centralized as the project grew larger so the "level of centralization is negatively correlated with project size, suggesting that larger projects become more modular, or possibly that becoming more modular is a key to growth". Crowston et al (2005) have specifically focused on the coordinating aspect of organizing. Their model builds on the collective mind idea of Weick and Roberts (1993) and in this way create a social focused model of coordinating but largely ignore technology as an actor. The view taken in this thesis is that no model of coordinating in open source can be presented without due attention to technology and the agency it has to organize both human and non-human actors. Our study then reflects organizing as a joint process in OS between all actors of the collective where neither human nor non-human is privileged.

Licence, Governance and Constitution

Recently there has been a renewed emphasis in the literature on the legal implications of OS addressing issues such as policies, licenses and public goods (Aigrain 2001, Aigrain 2002, Lerner and Tirole 2002a, Lessig 1999a, Lessig 1999b, Lessig 1999c, Lessig 2001). The license of a project can be seen as a critical concern because it is the blueprint of the collective and code, described as the Constitution of a project by Weber (2004, p179) "Yet there is another way to see the license, as a de facto constitution. In the absence of hierarchical authority, the license becomes the core statement of the social structure that defines the community of open source developers who participate in a project". This is reiterated by Stallman when he describes the GPL in terms of copyright of code, but also in terms of the inseparability of code and freedoms. "to copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code or any program derived from it but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable" (Stallman, 2006)[12].

Weber (2004) plays on this idea using the example of BitKeeper to show how there was a clash in Constitution between the GPL of Linux kernel software and the closed source license[13] of

[12] http://www.gnu.org/copyleft/copyleft.html

[13] **The Third Way of Licensing:** An interesting challenge to the governance and control in open source is found when the collective seeks to incorporate elements drawn from other jurisdictions and governed by other constitutions. For some such issues may be resolved by schemes of dual licensing. A dual license is what Olson calls the 'third strategy of licensing', it is not proprietary or open source, well not exclusively because it incorporates principles of both. Dual licensing is when companies "provide a single software product under two different licenses. One license, which imposes open source terms, is available to a certain class of users. A second license, with proprietary terms, is available to others" (Olson 2005, p71). All open source licenses are '"as is" licenses, with no warranties, no promises, and no recourse in the event of problems' (Olson, 2005, pp77) as are all open source strands of a dual licence. This may help companies mitigate risk. Olson (2005) sums up the main arguments in favour of dual licensing and provides reasons for why this form will

BitKeeper. The interesting element of this story is how both Constitutions were forced to live side-by-side [could some mixing be avoided?] and what effect this had on the governance structure of the Linux collective, collaboration patterns and importantly on the code created. Our findings from the case study expand on this in detail.

Moving on from a specific focus on the GPL Lessig focuses on the significance of code, "we live life in real space, subject to the effects of code. We live ordinary lives, subject to the effects of code. We live social and political lives, subject to the effects of code. Code regulates all these aspects of our lives, more pervasively over time than any other regulator in our life" (Lessig 1999a, p233). This sums up Lessig's (rather pessimistic and technological deterministic) view of control and regulation (Lessig 1999c) of the Internet and the world. Critics of Lessig's work claim that he has just added yet another metaphor to explain social ordering and that Lessig's "'code is law' [conclusion is a little] far-fetched" (Leenes and Koops 2005, p330) not to mention the "fact that the struggles in cyberspace amongst social orders are not fundamentally different from those that already exist in 'real' space social orders" (Savirimuthu 2005, p347).

Kiskis and Petrauskas (2005) also find dubious the idea that "technology, when tied to law, now promises almost perfect control over content and its distribution and it is this perfect control that threatens to undermine the potential for innovation that the Internet promises" (p307). Kleve and Mulder are even stronger in their criticism of Lessig's ideas and claim that "his theories are based on an outdated model of man and a misunderstanding of the role of technology. A more rational approach would be to accept that technological developments lead to social developments and to the evolution of norms and law... Firms will gradually move to better ways of making money. In the meantime, general levies for home copying could be arranged. Code is not law. At the present state of technology, if code would be law this could only mean Murphy's Law" (2005, p326).

Governance and Control

Many open source projects have been studied for their governance structures, but the bulk of such work focuses on a small subset, including Linux, Apache and Mozilla (Fielding 1999, Mockus *et al.* 2002). Mockus et al (2002) contrast Apache and Mozilla but both projects have some form of 'ruling committee' comprised of 12 or more developers. Their work on governance of open source projects has taken a strong functionalistic and narrow view. Authors such as Weber have gone beyond looking at governance simply in terms of functions and provide us instead with a

never stamp out either proprietary or pure open source licenses. Proprietary licenses will survive because companies will want and need to protect their heavy investments in new developments, and open source development and distribution will continue to thrive because it brings core infrastructure costs down. As Olson explains open source is innovative yet the most successful products are those that belong to technology which is stable, mature and easily commoditized like operating systems, databases and web servers.

stronger theoretical idea of governance as part of the Constitution of a collective that needs to find a formal accommodation. As Weber states it, there is always the ability to fork, and though "forks are rare but the threat of forks is a way to empower the community to have a strong voice" (Weber, 2005).

Weber (2004, p262) asks "what happens at the interface, between networks and hierarchies, where they meet?" Linux, as we show in the case study chapter, is not a typical network structure, nor is it a true hierarchy, rather it is a hybrid of the both, a netarchy as coined by Goodall and Roberts (Franck and Jungwirth 2002). Stewart and Gosain (2006) on the other hand discusses open source as clan based. Their model [which finds its roots in transaction cost theory] is based "on the idea that ideology provides clan control, which is important in OSS development settings because OSS teams generally lack formal behavioural and outcome controls". This is true, but the more you explore open source projects you realize that there are a significant number of rules and norms which if not followed make an outsider of that actor, no matter who it is. We even see, in Chapter 6, Torvalds pushed towards the periphery when he commits the sin of ignoring patches.

Control in OS, as Tuomi (2002, p173) proposes it "is indirectly based on capability to mobilize resources. Directly it lies very much with the users and potential co-developers". He then qualifies this by stating that, "The Linux developer community resembles a dynamic meritocracy where authority and control are closely associated with the produced technological artefacts", emphasising the power of technological reason (Tuomi, 2002, p173).

Thus governance studies can be traced to have focused on the role and functions of day-to-day governing to broader ideas of naming different governance structures to better understand them. We have the clan based hierarchy and even netarchy. Moving closer to the current study is Tuomi's point about control being in the hands of those developers that are closest to technology. Our study aims to push this yet further by dispensing with the human/material distinction so that a truer reflection of control and governance in open source collectives is allowed to emerge.

Conclusion

The aim of this chapter is to put into context the current study on learning and organizing in the Linux kernel collectif. The four main areas of literature that are scoped are technology of open source, learning and innovation in open source, organizing and coordinating in open source and finally license and governance modes in open source.

Various technologies are used for open source development but the focus in this work is on version control software. Such software, we claim, is able to mesmerize the collective that appropriates it in order to keep its true controlling nature under cover. This research studies the collective with version control as the central point of contact and the results revealed in Chapters 7, 8 and 9 prove quite intriguing.

The second area of literature dealt with here, learning and innovation in open source, marks the strong trend to date of focusing on innovation rather than learning, and that from an economic perspective. And the tendency to attempt to understand open source purely from an economic perspective is also evident in the third area tackled here, organizing and coordinating in open source. Often coordination is studied keeping only the human developers as the focus at the expense of technology. The current study draws learning and organizing together arguing that one is not fully understandable without the other. This is evident from Weber (2005a) explaining learning and learning distribution but he attempts to do so without explaining how such learning is coordinated or how the distribution of it is coordinated.

The final section of literature needed to put the current study into context is that on license, governance and questions of constitution. An open source process is regulated by the license which is used to create it. The license is thus the blueprint or constitution that governs the development process and collective (Weber 2004). The level of openness allowed by the license affects how learning and organizing occur in a collective.

Drawing all these points together the research question of this study emerges – *how does learning and organizing occur and manifest itself in open source collectives?* Version control software plays a role in both learning and organizing in open source collectives but the extent and detail of this role is what is explored in this work. A strong link between learning and organizing is drawn in order to understand both concepts better and then to understand how both occur in open source collectives. It is also clear now that licenses and governance structures of various collectives affect and direct how learning and organizing transpire so it is important to appreciate their significance. In the next chapter concepts of learning and organizing used in this research are explained in more detail where the inseparability of both is made more apparent.

Chapter 3
Learning to Organize a Collective

"Organizing and learning are essentially antithetical processes, which means the phrase 'organizational learning' qualifies as an oxymoron. To learn is to disorganize and increase variety. To organize is to forget and reduce variety" (Weick and Westley 1996).

Introduction

In this chapter we build the learning and organizing part of the framework needed for this work. Learning is addressed first. This is followed by consideration of organization and organizational learning. Finally the chapter considers the link between learning and organizing. As the title of this chapter suggests we seek to draw a strong link between learning and organizing, giving emphasis to the performative nature of creating and sustaining a collective. Both, it is argued, are needed to sustain the collective in some form, but it is a balance, or the performance of balancing the two, that keeps a collective alive. We argue, following Weick, that the noun form (organization, community etc.) denotes a form of stability, a stability that we believe is illusory. Our attention instead is on active, performative, organizing, and for the collective – collecting.

Background to Learning in Organizational Learning

This section positions learning within organizational learning literature. Cyert and March's (1963) book is considered to be 'the foundational work of organizational learning', but according to Easterby-Smith and Lyles (2003) it was Cangelosi and Dill (1965) who produced the first publication with "organizational learning" in the title. Learning does not have to be a conscious or intentional effort by individuals or an organization, and nor does learning necessarily increase the learner's effectiveness (Huber 1991). Rather Huber claims that an organization learns 'if any of its units acquires knowledge that it recognizes as potentially useful to the organization' (Huber 1991, p89). Senge's (1990, p3) definition of a learning organization captures more closely an open source collective, 'learning organizations are organizations where people continually expand their capacity to create the results they truly desire, where new and expansive patterns of thinking are nurtured, where *collective* aspiration is set free, and where people are continually learning to see the whole together'.

DeFillippi and Ornstein (2003) describe the dominant theoretical perspectives in organizational learning based on psychological explanations. Among them are behavioural theories of learning defined as 'those theories which focus on the antecedents to and changes in organizations' routines and systems as the organization responds to its own experience and that of other organizations' (DeFillippi and Ornstein 2003). Nelson and Winter's (1982) 'evolutionary model of the firm' is quoted by them to be the most well-developed theory of behavioural learning. Their use of two main concepts of path dependence and trajectory combine to explain how history is important in shaping the future form of an organization and how it learns and develops in a certain direction or trajectory of cumulative learning.

In contrast, social construction perspectives consider that 'learning is embedded in the relationships and interactions between people' (DeFillippi and Ornstein 2003, Orr 1990a, Orr 1990b, Wenger 1998). Penrose (1959, p52), because she believed that 'success depends upon a gradual building up of a group of officials experienced in working together' was essentially proposing a social constructionistic model of learning. Social constructionists thus 'emphasize that organizations know more than the sum of the knowledge of individuals within them' (Easterby-Smith and Lyles 2003, p8) and at the same time they also, like Penrose, emphasize the significance of experience.

From this perspective learning is considered to be a social process which is based on concerted (collective) sense-making (Weick 1991). Both Nonaka and Takeuchi's (Nonaka and Konno 1998, 1995) SECI model and Lave and Wenger's (1991) community of practice models of organizational learning are based on Weick's sense-making ideas. The community of practice theory emphasizes how sharing a common language, practices and values can lead to the formation of a learning community where newcomers become part of the community through a process of legitimate peripheral participation (Lave and Wenger 1991). But as Hargadon and Fanelli (2002, p300) clarify, '"shared knowledge" refers not to knowledge that is commonly held (meaning identical) across technicians but rather knowledge that *has been shared* between technicians—knowledge made empirical through the generation of social artifacts, in their case stories, and made latent again through each individual's interpretation of those stories'. We can then agree with Simon (Simon 1991, p125) and others that, "all learning takes place inside individual human heads," just as we agree with Boland and Tenkasi (1995, p335) that "the individual does not think in isolation and is not an autonomous origin of knowledge"'. Nonaka and Takeuchi (1995) similarly understand learning to occur in collectives but they speak more about teams where the members pool their resources and ideas and through discussion come to new ideas and understanding. Weick and Westley's (1996) emphasis, as ours, is the collective organizing and they call it the collective mind because they believe that this phrase pays attention to action, "the word "collective" unlike the words "group" or "organization", refers to individuals who act as if they are a group. People who act as if they are a group interrelate their actions with more or less care, and focusing on the way this interrelating is done reveals collective mental processes that differ in their degree of development. Our focus is at once on individuals and the collective, since only individuals can contribute to a collective mind, but a collective mind is distinct from an individual mind because it inheres in the pattern of interrelated activities among many people" (Weick and Roberts, 1993).

Pentland's (1995, p2) framework, the 'organizational knowledge system', in this tradition views an organization 'as a social knowledge system... within which phenomena like organizational learning can be analyzed and interpreted'. He grounded his framework in the sociology of knowledge taking ideas from Holzner and Marx (1979), Berger and Luckman (1966), Bloor (1976) and Latour (1987). The idea of his framework is to analyze organizations as '"knowledge systems" composed of a collection of socially enacted "knowledge processes"' and this framework is limited for analysis of organizational learning because 'it focuses on pragmatic knowledge that is intended to achieve a certain end within a certain time and space' (Pentland 1995, pp2-3).

Weick (1991) makes the assertion that learning is a topic that seems to have engaged the interest of organizational theorists at a point where psychologists (behavioural perspective) have concluded that it is not a workable concept. Weick (1991) makes a simple learning definition the centre of his argument and then works around it to suggest that perhaps it is not productive to disregard old definitions as they provide the building blocks for future work. In psychology learning is defined as a "combination of same stimulus and different response" (1991, p117). There are two strategies possible, according to Weick, the first is to stick to this traditional definition and to understand organizational learning in light of it or otherwise to look elsewhere for a new definition. He weighs the advantages and disadvantages of both strategies and finally concludes that the first strategy is probably the better. One out of the many arguments he develops to make his point is the distinction between organizational learning and individual learning because the "combination of same stimulus, different response is rare in organizations meaning either that organizations don't learn or that organizations learn but in non-traditional ways. Choice between these two possibilities has important consequences" (1991, p117).

The distinction made in learning literature between the terms 'organizational learning' and a 'learning organization' also needs consideration. *Organizational learning* 'refers to the study of the learning processes of and within organizations' (Easterby-Smith and Lyles 2003) whereas a *learning organization* is 'seen as an entity, an ideal type of organization, which has the capacity to learn effectively and hence to prosper' (Easterby-Smith and Lyles 2003) or as Tsang (1997) adds, it refers to a 'particular type of organization in and of itself'. This important debate is picked up by Jones (1995) where he fractures the literature on organizational learning and a learning organization into categories where organizational learning happens independently of individuals, or through individuals. Jones stresses that in order to better make sense of organizational learning researchers must adopt approaches that "recognize the social nature of individual cognition and the situated nature of learning" (Jones 1995, p75), and also that applying this understanding to information systems design could lead to more effective systems.

Senge's (1990) work focuses on dialogue in an organization arguing that it is through communication or conversation that understanding is achieved. Gadamer (1975, cited within Senge (1990)) explained this idea of dialogue/conversation in his work *Truth and Method.*

> "Thus it is a characteristic of every true conversation that each opens himself to the other person, truly accepts his point of view as worthy of consideration and gets inside the other to such an extent that he understands not a particular individual, but what he says" (Gadamer 1975, p347).

It is then not about agreeing *per se*, but about having the conversation which leads to better understanding. In this way we may see open source developers use of chat forums and email. This 'conversation' allows them to mould their ideas and may lead to innovation in process as well as product. In open source, for our research, the process and product cannot be fractured apart, they are often one and the same, or better still one so effortlessly steals into the other that to study any one alone would be both difficult and unnecessarily reductionist.

With many learning theories to choose from in this work we have returned to a basic concept of learning offered by Bateson. As explained below this simple yet powerful definition of learning can be effective. Bateson focused on human learning but his definition can be extended to include collective learning.

Bateson's Model of Learning

"[Man] appears to us ... as a creature who *learns*."
(Bateson quoted in Visser (2003, p270))

Gregory Bateson extended the cybernetic model introduced to him by Norbert Wiener showing interdependence between humans, nature and technology. This provided a stimulus for many in the anthropology field to take this understanding of social systems further, including Haraway's 'cyborg manifesto' and Bruno Latour's work on 'hybrids' (Tresch 1998). Bateson focused on '*interactions*' rather than upon fixed structures and this influenced his work on learning. Learning is not a process or experience which comes about in isolation, for Bateson, it involves interacting with the environment and others and is about 'shared understanding' (Tognetti 1999). Most free/open source developers do not work in isolation. It is the process of peer review and brainstorming which is carried out over the message forums and email that leads to a kind of shared understanding which then leads to the creation of software and software process. Software versions are one reflection of this, but so too is the ongoing performance of the collective. Bateson's theory is essentially a behavioural theory of organizational learning, as is Weick's (1991), and engages two concepts, that of deutero-learning and double bind, explained below (Visser 2003). As a behavioural theory it allows one to focus on 'observable interactions' (Visser, 2003) which form some reflection of a level of learning. However, this does not imply that a social understanding of collective learning cannot be extended to both.

Bateson (1972, p283) gives a simple definition of learning as 'a *kind* of change'. 'Change denotes process' (Bateson 1972, p283) but he explains that a process itself is 'subject to "change"'. He elaborates his ideas of learning *as* change through the metaphor of acceleration. He lays the foundation for various levels of learning that he believes can or do exist. These are 'logical categories' of learning from Level Zero onto Level III [and perhaps Level IV]. An individual evolves, through contact with the environment and self-reflection, from one level to the other. Bateson's levels of learning are based on individual learning [through interaction with others] but Level IV, which is evolutionary change, focuses more explicitly on what we might call organizational learning (Rothwell 1983).

Star and Ruhleder (1996) use Bateson in their infrastructure study of the Worm Community System. This work provides a good example of how effective and useful Bateson's levels of learning ideas are for studying human/technical collectives. Though Bateson's ideas on learning form the basis of some influential learning literature today, it is strange how little research has been carried out in this area using his ideas. The infrastructure study by Star and Ruhleder (1996) stands out in its application of Bateson's levels of learning. Most related work found by this author was in the management area and application was more poised towards how to better manage employees, which does have possible significance even for the current study, but it still doesn't help to understand why organisational learning literature bears little stamp of Bateson. On the other hand knowledge management researchers make heavy use of Argyris and Schon's work which they admit themselves is heavily based on Bateson.

Bateson's Levels of Learning

Zero Learning

Zero learning does not mean that there is no change, but that there is enough change to maintain a 'constant velocity'. So in order to keep the status quo the change required may be very minimal but that in itself is enough to merit zero learning. Zero Learning involves 'the receipt of a signal...

not subject to correction by trial-and-error' (Bateson 1972, p248). To use Bateson's acceleration example, this type of learning could be explained as constant velocity. Zero learning is defined as 'the case in which an entity shows minimal change in its response to a repeated item of sensory input' (Bateson 1972, p283). Bateson supplies us with a number of examples which reflect this level of learning, like that of habituation in an animal, where it stops 'to give overt response to what was formerly a disturbing stimulus'. It is a very primitive form of learning which includes animal instinct (Rothwell, 1983).

Learning I – Proto-Learning

Learning I is the next step in the hierarchy. Bateson sees learning as change and that implies, as a premise, a 'notion of repeatable context' (Bateson 1972, p292). This, in *Steps to an Ecology of Mind*, is explained with the Pavlov conditioning of the dog example. At Time 1 Pavlov rang a buzzer and then accompanied that with some food. At Time 2 he just rang the buzzer and the dog began to salivate in anticipation of food, even though no food was given. The context was the same [as much as possible] so the buzzer became the stimulus for another response. The same can be said by using the example of rote learning, 'in which an item in the behaviour of the organism becomes a stimulus for another item of behaviour' (Bateson 1972, p288). Bateson uses the concept of a 'context marker' to reflect how 'an organism responds to the "same" stimulus differently in differing contexts' where context is 'a collective term for all those events which tell the organism among what *set* of alternatives he must make his next choice' (Bateson 1972, p289). This level includes memorization and other instrumental change which is adopted to either gain a reward or to avoid punishment (Jacobson 2003).

Learning II – Deutero-Learning

Learning II is a change in Learning I. It has also been called 'deutero-learning' and 'learning to learn'. It, according to Rothwell (1983, p24) is a 'discovery of how one learns best'. Bateson continues with the Pavlov example to explain how Learning II differs from Learning I. If an individual is capable of showing the same behaviour in a different context then this reflects an ability to adapt and assimilate or Learning II. Take rote learning as an example, an individual, if he continues to practice learning any material in this way, will deepen his experience and will begin to rote learn better, faster and more efficiently. This reflects that he will have devised context markers for himself that will stimulate his mind to recall more material. But what will really differentiate this from Learning I is the way this individual has reflected on his learning so far and devised a scheme or technique to improve his ability to learn. Bateson (1972, p301) adds that, 'this self-validating characteristic of the content of Learning II has the effect that such learning is almost ineradicable'. We are also given the example of "reversal learning" where meanings given to something are reversed. If the individual is able to not only spot the reversals but can detect a sequence in the reversal pattern or 'contradictions', then Learning II can be said to have occurred.

Learning III – Trito-Learning

Learning III, according to Bateson, rarely occurs in human beings. This requires a 'profound reorganization of character' which may sometimes be found in religious conversion (Bateson, 1972, p301). This could be an attempt by an individual to 'resolve the contraries' raised by Learning II in order to improve learning. Indeed Bateson (1972, p306) explains Learning III with a comparison to psychosis where individuals 'find themselves inhibited from using the first person pronoun'. However, in this work, focused on the collective, the potential for 'profound reorganization' may be more possible, for example, as in the case presented here, a transition away from total commitment to the open source license.

Learning IV – Organizational Learning

Bateson never presented a Level IV of learning. Rothwell (1983), however, suggests that Bateson's learning concepts can be connected to theories of open systems and organizational learning. He draws on Argyris and Schon (1978) and argues that Learning IV might be understood 'as an organization's ability to learn through time' (Rothwell, 1983). Learning III forces individuals to assess 'fundamental assumptions about how they learn' but Learning IV suggests how the same would happen for organizations which would imply a serious shift in organizational culture which is sustained over time' (Rothwell, 1983).

In the many interpretations of Bateson's work the common explanation of the levels of learning is an interesting phenomenon, in spite of the fact that most of the researchers belong to different fields (Burgoyne and Hodgson 1983, Wills 1994). One of the commonalities between the interpretations is the lack of mention of zero learning. Most authors have been rather dismissive of this level. *Learning 1* is about short-term learning whose relevance is immediate. This sort of learning makes little impact on the personality of the individual. *Learning 2* is about behaviour change but not about drastic personality change. This behaviour change is a result of a building up of knowledge, and the interesting role of this behaviour change is that it is transferable from one situation to another. *Learning 3* is related to ideas of self-awareness. When an individual becomes conscious of his own behaviour and learning pattern then he is more capable of changing it - 'learning how to learn'. This level of learning is consequently not situation-specific. Wills (1994) renamed these three levels respectively as cerebral learning, behavioural learning and transformational learning. These names reflect well the difference between the various levels. The claim made is that all learning [with the exception perhaps of zero learning] is stochastic, meaning it 'contains components of "trial and error"' (Bateson 1972, p287). Thus each level of learning is about a process of correction of past errors and learning enough about why they happened in order to improve the ability to learn. Weick's sense-making is also contingent upon reflection in hindsight.

Deutero-Learning and the Double-bind

Rothwell (1983) understands Bateson's theory of learning as a heterarchy[14] which is based on three assumptions.

- Learning *denotes change*, but the degrees and the types of change may differ.
- Learning is a *communicative process*, but the degrees and the types of communication may differ.
- Learning involves the *mastery of new approaches or solutions* to problems (Rothwell 1983).

Bateson linked the idea of message levels [metacommunication] and paradox to develop the concept of deutero-learning. However, in the case when "Type confusion leads to paradox when both message and meta-message contain negatives. On this principle we can imagine the generation of paradox in the deutero-learning system when an organism experiences punishment following some failure and learns that it must not learn that punishment follows failure" (Bateson in a letter in 1954 quoted in Lipset (1980, p205)). This is what Bateson calls the 'double-bind' or "pathological deutero-learning" (Bateson 1963, p180). If a mother punishes her child for a particular behaviour like screaming but then also punishes that child for *learning* that punishment will follow the act of screaming she then 'induces a paradox' in him because she has 'combined negative proto-learning with negative deutero-learning' (Visser 2003, p272). A sustained and prolonged exposure to such double-bind communication and learning, especially when the child cannot escape from this situation or speak out his mind will, according to Bateson, lead to schizophrenia (Lipset 1980).

Bateson explains *metacommunication*, which can be useful when considering how interaction is not just based on words or text but rather on other physical cues as well. This 'communication about a communication' has three types: 1) messages which are 'mood-signs', 2) messages which stimulate mood-signs, and 3) messages which enable the receiver to distinguish between mood-signs and any other signs which resemble them (Bateson 1972, p189). In open source interaction

[14] The word heterarchy is formed from the Greek prefix *heter*, meaning different, and the Greek suffix *arche*, meaning to rank. Hence heterarchy is a ranking by differences or contrasts as opposed to hierarchy which is ranking based on importance (Rothwell, 1983).

'smileys' and various emoticons which are used abundantly provide the role of metacommunication.

Building on Bateson

Argyris and Schon (1996) base their organizational learning work on Bateson's (1972) levels of learning and Ashby's (1940) distinction between single-loop and double-loop learning [see Figure 4]. Drawing on Bateson's idea of maintenance of constancy in a living system as indicative of the kind of change required in order to sustain organizational constancy, Argyris and Schon (1978) call such organizational maintenance 'zero-order learning' (Schon 1975, Schön 1983, p119). "Single-loop learning is the error-detection-and-correction process… Double-loop learning occurs when error is detected and corrected in ways that involve the modification of an organization's underlying norms, policies, and objectives" (Argyris and Schön 1978, pp2-3).

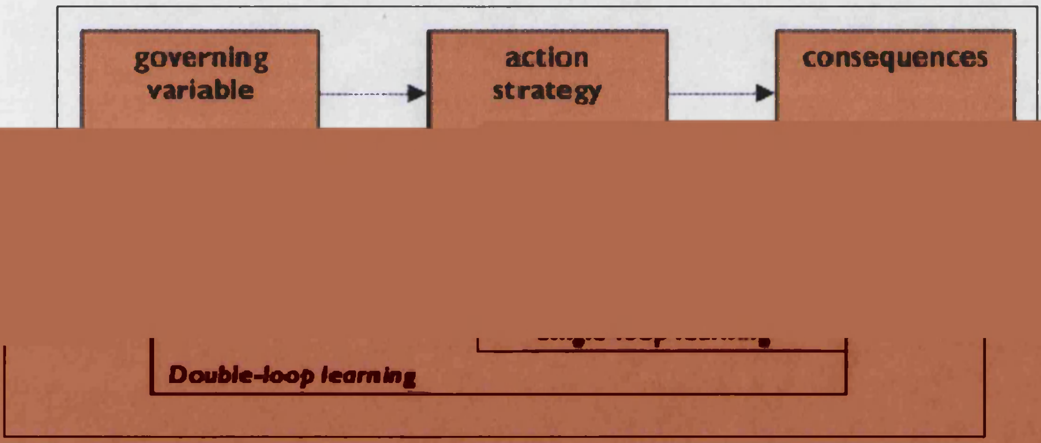


Figure 4: The model and three elements of theory of action (Argyris 1982, Smith 2001)

Argyris and Schon (1978) draw on Bateson more heavily when it comes to *deutero-learning*. This would correspond with Bateson's second-order learning and can be described as the process of 'learning to learn' or as Argyris and Schon claim, learning how to single and double-loop learn. "When an organization engages in deutero-learning, its members learn, too, about previous contexts for learning… They discover what they did that facilitated or inhibited learning, they invent new strategies for learning, they produce these strategies, and they evaluate and generalize what they produced. The results become encoded in individual images and maps and are reflected in organizational learning practice" (Argyris and Schon, 1978, p27).

Constituting an Organization

Argyris and Schon (1978) explain what distinguishes a collection or mob of people from an organization. An organization implies that members develop measures which firstly, allow members to make decisions on behalf or in the name of the collective; secondly, delegate to certain people the authority to act for the collective; and lastly, to set a boundary between the collective and the 'rest of the world', thus creating a well defined 'we' feeling (Argyris and

Schon, 1978, p13). These measures or rules developed by the members may be tacit but must have some form of continuity to ensure that the organization continues to exist even as its members leave or new ones join. The three conditions laid out above provide some indication that the case being researched, the Linux kernel development, could indeed be an 'organization' - it qualifies on all three points. There is a boundary between the community and the rest of even the open source world. This boundary is most noticeable when an individual wants to join this organization. Joining is not as easy as it seems. You have to prove your ability as a developer and a debugger. And many times, you begin as a debugger and then work your way up from a peripheral membership to developer status and beyond (Zhang and Storck 2001).

An organization, in these terms, is made up of individuals so how does one distinguish between individual and organizational learning? To begin with, an organization 'is not merely a collection of individuals' and organizations often 'know *less* than their members' (Argyris and Schön 1978, p9). In a later work, Schon (1983) elaborated on individual versus organizational learning through an example of a craftsman making wooden shovels. To describe individual learning Schon explained the steps and chores the craftsman must perform from log selection right down to the last smoothing of the wooden shovel. If the craftsman would not be able to find a particular type of wood that he is used too working with then he may, on occasion, switch to another. This reflects how he would adapt and amend his 'theory of action' which reflects how he learnt and changed. Would an organization then consist of a number of craftsmen performing the same tasks? This is where Schon distinguishes between an individual and an organization learning. In an organization the individuals would allocate themselves to various duties according to expertise or other factors and there would thus be delegation and division of labour. So, it is *agency* that is important, '*just as individuals are the agents of organizational action, so they are the agents for organizational learning*' (Argyris and Schön 1978, p19).

Organizational learning is then defined by Argyris and Schon (1978, p19) as learning which occurs "when individuals, acting from their images and maps, detect a match or mismatch of outcome to expectation which confirms or disconfirms organizational theory-in-use". In order for organizational rather than just individual, learning to occur all detections and changes must be 'embedded in organizational memory' because if they aren't then when the individual leaves the organization he will take the learning with him, thus 'individual learning is a necessary but insufficient condition for organizational learning'. Attewell (1992, p6) argues likewise when he says that "the organization learns only insofar as individual skills and insights become embodied in organizational routines, practices, and beliefs that outlast the presence of the originating individual."

Organizational Memory

Individual members of the organization have to devise a way to 'measure' the quality of their work according to some criteria set by them and the rest of the organization. Schon (1983, p117) calls these 'organizational artifacts' which consist of maps, programs and memories. A *map* is a 'picture of the system – organization charts or building plans. *Programs* describe sequences of organizational action in work-flow diagrams... or in lists of procedures. *Memories* are reservoirs of information about past organizational experience, for example, price or inventory lists: they may be kept in publicly accessible files or in the head of an "old hand"'. The Linux kernel FAQ is a simple example of a *map* used by the developers as guidance for communicating their work and questions online.

Individuals leave an organization and remove learning, "In order to avoid such "organizational entropy" or loss of organizational memory, members must continually work at the recruitment and instruction of new members, the preservation of organizational maps and memories, and the detection and correction of errors" (Schön 1983, p119). This scenario is very real for many open source projects. The initial stage of any project is extremely difficult because there is no guarantee that it will manage to garner enough support from the larger community of developers or keep their attention, 'volunteers are not cheap so code is not cheap – so it is kept archived' (Massey 2003).

Another form of organizational memory in an open source project is the archive kept in the version control system studied here. Most patches created undergo peer review and some amendments. Version Control keeps a track of every change, who made it, why it was made and exactly what kind of a change it is. In order to get a more complete picture of the software under development and to assess how it is has evolved an individual has only to backtrack to all the small changes, called deltas, to piece together a full picture. SourceForge.net is a website which performs the key role of an infrastructure and organizational memory for thousands of open source projects, 'having an organizational memory is one of the top 3 selling points of SourceForge' (Augustin 2003).

Walsh and Ungson's (1991) seminal paper on organizational memory provides an interesting overview of how organizational memory is seen and defined by various academics. They explain how Argyris and Schon (1978) consider organizational memory to be a mere metaphor because "organizations do not literally remember", in contrast to Sandelands and Stablein (1987) who define organizations as "mental entities capable of thought". Walsh and Ungson (1991, p59) relate how other opinions 'fall some place between these rather divergent perspectives' and it is 'unclear as to whether information is stored and processed by individuals who comprise the organization

(Kiesler and Sproull 1982, O'Reilly 1983, Sims and Gioia 1986, Ungson *et al.* 1981), by the organization itself (Galbraith 1977), or by the dominant coalition or upper echelon as a reflection of the organization (Hambrick and Mason 1984)'. These authors do, later in this paper, clarify that it is important to study the individuals of an organization and how they acquire, store and retrieve information in order to be able to understand how organizations learn and acquire a memory. They make this claim based on the simple argument that it is individuals which comprise an organization.

Alavi and Tiwana (2003) understand organizational memory as 'stocks of organizational knowledge' and include storage and retrieval as necessary characteristics. They comprehend organizational memory to include an internal and external part. Internal memory is 'the stocks of knowledge that reside within the individuals or groups of individuals in an organization' and external memory 'contains codified and explicit organizational knowledge and includes formal policies and procedures, and manual and computer files' (Alavi and Tiwana, 2003, p108). Duncan and Weiss (1979) understand that a 'facility' must exist within an organization which is capable of holding 'communicable, consensual and integrated knowledge'. One view of organizational memory is that a 'clear view of the past will obscure an accurate view of the present' (Walsh and Ungson, 1991, p72). However, Hedberg *et al* (1976, p41) provide an opposing view that, 'footholds in time are the appropriate components for assembling trajectories into the future'. We understand open source collectives to have managed a healthy balance between avoiding getting stuck in old patterns (in other words such collectives have a healthy approach towards unlearning) and also have the ability to build on prior work. Version control software holds repository of code over time so there is definitely a strong history, however, when you access the code you realize that each code comes forth from various trajectories and that code is a building block that can be teased apart to create something totally new. Thus Raymond's (1998) "*Good programmers know what to write. Great ones know what to rewrite (and reuse)*".

We can assume that open source developers are not often in face-to-face contact so there is a strong need to keep information stored in a way that is retrievable by all, and more than one of them could be working on the same problem at one time without being aware of each other. This is where version management tools play an important role in not allowing an overwrite of any patch or 'knowledge creation'. Repositories 'bring together content from various data sources, providing a unified access point and reducing knowledge search costs' (Hansen 1999). The Linux email archive is the kind of repository that von Krogh would describe as a 'communal resource' where knowledge is shared 'without the prerequisite of an agency that monitors and enforces cooperation' (von Krogh, 2003, p379). A view which we challenge here.

As we learn later, when the Linux kernel collective debates over version control software adoption is described in more detail (Chapter 6). Indeed, one of the key ideas to emerge amongst the various issues that were argued was the strong controlling force such software exercised over the collective.

Huber (1991) concluded that there are four variables that can influence organizational memory, namely, membership attrition, distribution and interpretation of information, norms and methods of information storage and finally, methods of locating and retrieving information from the memory stores. Membership attrition is quite a concern with open source communities. Of course, to obtain a critical mass of developers is the primary concern of any new project but then to be able to maintain a certain level of membership becomes the long-term concern. Benkler (2002) describes how, through a fine granularity of modules in open source projects, this becomes easier to manage because no one developer is over-burdened and many are free to join the community fleetingly, add their tiny patch and then disappear. If enough developers behave this way then the project will continue to grow because this 'allows the project to capture contributions from large numbers of contributors whose motivation level will not sustain anything more than quite small efforts towards the project' (Benkler, 2002, p10).

The interesting quality of open source communities is that there is a conscious effort to store information. The developers communicate via email and these email messages are all archived for future reference. Fogel (2005) when speaking of mailing list archives claims that "archives are precious data – a project that loses them loses a good part of its collective memory" (pp59). Even the IRC conversations that developers engage in are saveable but perhaps they are not accessible to more than just the individuals who were party to the real-time discussion. The latter feature could well fall into Huber's (1991) claim that certain kinds of memory are difficult to store or are easily erased, like wiping a blackboard clean after a class. This happens in situations when people are unaware of the importance of what they have discovered or written, or when they are unable to anticipate a need for this information in the future. IRC chat logs, if not loaded onto a shared site or server, are hard or impossible for other developers to locate and access.

Unlearning

We have argued here that learning relies on memory but what about unlearning? Martin de Holan and Phillips (2003, p393) state 'that forgetting is the necessary counterpart of learning'. Their model of organizational knowledge links organizational memory to learning and forgetting. Hedberg (1981, p18) was one of the first to write about this concept and defined unlearning as 'a process through which learners discard knowledge' and this knowledge is supposedly 'obsolete and misleading' (p3). As Huber (1991) points out Hedberg's definition has a strong implication that this knowledge is useless and outdated and so to be rid of it 'is functional' (Huber 1991, p104). This is a rather narrow view and Huber attempts to rectify this by allowing for the idea that unlearning may be a disadvantage to an organization. He discusses how unlearning 'can lead to either a decrease, or an increase, in the range of potential behaviours' (Huber 1991, p104). As Martin de Holan and Phillips (2003) indicate there has been much work on learning and organizational knowledge management but forgetting and unlearning has only been touched upon in passing by a few like Bettis and Prahalad (1995), Day (1994) and Nystrom and Starbuck (1984), Peters (1994) was particular about mentioning that unlearning is an area in dire need of further research yet offered little more than a short vignette on the issue. Depreciation of knowledge is a premise offered by Argote (1999), where depreciation was explained as when 'products or processes change and render old knowledge obsolete... organizational records are lost or become difficult to access... [due to] member turnover'.

Four effects of unlearning were proposed by Huber (1991) and they increase the scope of what it means to unlearn. *Firstly,* if for whatever reason, an organization has been through some unlearning then it is very likely that there will be a time period when it will become inactive in the context of where this knowledge was previously used. *Secondly,* with the right incentive to find a substitute fact or belief, the organization would make every effort to accomplish this while still maintaining other functions. Here Huber points out that certain authors like Lindblom (1959) and Cyert and March (1963) believe that often this search for a replacement belief leads the organization to something very close to the recently unlearned belief as the search is focused 'in the same vicinity' (p104). Of course there is a potential problem with this 'solution' because if the first belief was unlearned for the reason that it was not effective or beneficial then finding a close substitute as a replacement would hardly do good. *Thirdly,* unlearning can have the positive effect of opening the way for 'new learning to take place' (Huber 1991, p105). One very extreme form of this type of unlearning in an organization is cited as a discharge of employees who are not considered to be able to unlearn old ways and adopt the new. And *lastly,* unlearning can be said to be part of the socialization of new members in an organization by forcing 'the knowledge that the new members possessed upon entry [to] become unavailable to the organization' (p105).

Having explained what is learning (Bateson 1972), organizational memory (Walsh and Ungson 1991), and unlearning (Huber 1991, Martin de Holan and Phillips 2003) we now draw the link between learning and organizing (Clegg *et al.* 2005).

Organizing and Learning

"*Knowledge of organizing and the organizing of knowledge implicate and explicate each other and are thereby inextricably intertwined*" (Chia 1998).

The main argument in this chapter is that organizations exist *through* learning. It is the process of learning that creates and sustains. Ontologically, this understanding complements that of Actor Network Theory [more on this in Chapter 4], where ideas of performativity and reality creation are interlinked. Organizations are described by Hasselbladh and Theodoridis (1998) as 'magmatic mode(s) of being' because they are never stable and nor do they take any specific shape, but are always in flux. Organizations never take any specific shape and we understand this to mean that they never stay the same for long enough to be recognized in one particular form. According to Clegg *et al* (2005) this constant change is reflective of an organization learning through the continual acts of organizing, 'learning is thus a form of dis-organization that connects with and can destabilize the desire for a unified, timeless and static idea of organization' (p161). Organizing is the movement to stability, formality and structure, while learning is the movement into or towards the unknown, to chaos or instability. Too much of either cannot be good for the long term prospects of any organization or collective but too little may be equally fatal. Thus, Clegg *et al* (2005) argue, in the tension between the dimension of organizing (closing down discussion, establishing boundaries and excluding, resolving issues, managing information flows), and that of learning (opening up discussion, engaging with and including the new, absorbing variety and encountering new information sources), we can seek the generative processes that sustain and mutate a collective over time. And 'organization is the knot, the fold, where order and disorder meet. It is the very process of transgressing the boundaries between the old and the new, the stable and the unstable' (Clegg et al. 2005).

Becoming is not a specific state but rather a focus on movement from the then to now, not a move from one state to another. It reflects a passing of time and a process, 'becoming thus sees the idea of an organization's existence not as an ontologically stable, but rather as something that exists only in its duration' (Clegg *et al*, 2005, p159). It is in this becoming that organizing and/or learning occur, and 'considering learning in terms of becoming focuses on movement rather than that which is moved' (Clegg *et al*, 2005, p159). More specifically learning 'is the driving force beyond organization' and a necessary precondition for learning is organizational slack. Chaos and disturbance, and in particular engagement with those outside, demands a response which opens up the collective to future possibilities; it is in this uncertain and scary 'movement towards' some not quite known future, that is 'becoming' – not certain, not defined, but a journey into the chaos.

Organizing and learning are thus constitutive of becoming; played out in the spaces or folds between an extant order (that which is already organized) and futures yet to be encountered.

The previous tradition of studies of change have been criticized for focusing on stability in order to understand change (Feldman 2000, Orlikowski 1996, Weick 1998). This suggests the need to reverse 'ontological priorities' (Tsoukas and Chia 2002) and for keener perceptions of the ongoing nature of change or 'changing' (Weick and Quinn 1999). This reversal is helpful not least if it allows a better understanding of the micro-processes of change, treats change as dynamic and unfolding rather than as a fait accompli, and makes it ontologically possible to 'see' change by directly looking for changing, rather than as a byproduct of some comparative stabilities (Tsoukas and Chia 2002). Thus in developing the concept of becoming Clegg et al. (2005) emphasize the focus on movement, not on what has moved or where it arrives (at best mere snap shots, moments in time); becoming is about travel and mutation rather than what has mutated. Stability is then at best fleeting but more likely to be illusionary; change is reality.

Becoming is then a way of renaming and revealing the arena where learning and organizing resolve and shape the futures of the collective. We speak of futures in the plural because becoming is essentially the journey, not the destination, and just as a ship sails 'towards' a port, not 'to' a port, the collective can at best debate and resolve on a new course to steer, but can never arrive (Orlikowski and Hofman 1997). Becoming is thus an essential space of connections that can express the dynamics of learning and organizing. Becoming is not the specific state that is left, nor what is achieved, but the movements that take us from 'then to now', or now towards some future, embodied in the tension between outward looking and exploratory processes of learning, and cautious and considered acts of organizing. This duality has to coexist in order for either to operate. A fine balance that Clegg et al (2005) describe as a 'generative dance on the edge of a volcano'.

Other authors have discussed this as the 'space' of learning (Nonaka and Konno 1998). They argue that knowledge is created and fostered in 'shared spaces' called *ba.* Ba is a Japanese concept established by Kitaro Nishida, however, Nishida never linked *ba* to knowledge creation. Nonaka and Konno (1998) used the concept of *ba* along with their ideas on tacit and explicit knowledge to show how individual tacit knowledge can be made, first explicit individual knowledge, and finally, organizational tacit knowledge.

Many authors (Clegg *et al.* 2005, Patriotta 2003, Weick and Westley 1996) have drawn on Bateson's idea of breakdowns and the belief that learning or understanding occurs after action, so hindsight is needed to better understand the conditions under which learning occurs or becomes necessary (Argyris and Schön 1978). A breakdown, as a catalyst for learning, is a very useful concept because it helps to operationalize learning. As a researcher curious about learning, it helps to focus in on something as elusive as the moment when learning can be said to occur. The core of Weick's (1988) idea of sensemaking is that 'action precedes cognition and focuses cognition' (Weick, 1988, p307). We act first and then make sense of what we have achieved in hindsight or in the process of acting. Learning thus occurs in doing. Actor network theory understands this idea well and builds on this concept of action in doing, its ontology being, in terms of action between more than one actors creating reality in the process of their interaction. Action is organizing and organizing is also called interrelating (Weick and Roberts 1993). Indeed, without making the link explicit, the authors combine organizing with learning – they call it heedful interrelating – where heedful encapsulates 'an important set of qualities of mind' (Weick and Roberts, 1993, p361) and people are heedful when "they act more or less carefully, critically, consistently, purposefully, attentively, studiously, vigilantly, conscientiously, pertinaciously" (Ryle 1949, p151 - cited in Weick and Roberts (1993)). Heedful acts imply that the 'agent is still learning' (Weick and Roberts, 1993, p362). Heedful interrelating implies purposeful organizing where something is being learnt. As Weick and Roberts (1993) emphasize, who is doing the learning, the individual or the collective is still an open question, "learning is not an inherent property of an individual or of an organization, but rather resides in the quality and the nature of the relationship between levels of consciousness within the individual, between individuals, and between the organization and the environment" (Weick and Westley 1996, p446).

"Learning is an ongoing and implicit feature of the organizing process. By this we mean that as organizing unfolds, it does so in ways that intermittently create a set of conditions where learning is possible. We call these 'learning moments'. As organizing becomes disorganized, the forgotten is remembered, the invisible becomes visible, the silenced becomes heard. These changes create an opportunity for learning. Learning can be said to occur when forgetting, concealing, and silencing hide a *new* set of conditions and in their place create new categories, different meaning, and more organization" (Weick and Westley, 1996, p456). Part of the ability to organize and learn depends on the ability to unlearn. Weick (1996), using the example of the fire-fighters of the Mann Gulch disaster, explains how in many cases unlearning is required to cope or organize in a particular situation. He stresses that there are a number of reasons why unlearning can be difficult, even if lives depend on it. Some such reasons are a lack of trust in leadership, loss of control, dilution of intensity of crises because of fading social dynamics, and skills to generate replacement activities.

The question arises then as to how one can study the spaces where learning occurs because "Learning moments and spaces tend not to be obvious precisely because they retain vestiges of order, routine, and expected exploitation. They are *almost* business as usual" (Weick and Westley, 1996, p451).

Conclusion

The key concepts addressed in this chapter are learning and organizing. We adopted Bateson's definition of learning as change. This definition is broad and flexible enough to encompass all manner of actors, both human and non-human. Drawing on Weick this study appreciates that learning and organizing are interlinked concepts and the study of one often leads to the exploration of the other. Organizing indicates the tendency towards stability, however short-lived it is eventually. We emphasize the link between learning and organizing through the use of Clegg *et al.'s* (2005) becoming concept, where becoming helped to bring the focus to the actual process that unfolds into learning and/or organizing.

Two more issues need to be carried further in this thesis, the importance of organizational memory and the concept of unlearning. Organizational memory has real significance for both learning and organizing as it holds together both the past and future trajectories that are possible. The paths that can be taken are usually heavily dependent on what we already know and understand. Something new usually emerges from the familiar [though not always as revolutionary innovation would otherwise not be possible]. The existence of an organizational memory can also make it somewhat difficult for an organization to experience dramatic and perhaps needed change. This sort of change [Level IV learning] requires unlearning in order to learn afresh. Unlearning or forgetting is then the necessary counterpart to learning (Martin de Holan and Phillips 2003), the catalyst for innovation.

Chapter 4
The *Performative* Ontology of Actor Network Theory

In this chapter Actor network theory, with its distinct ontology, was chosen for its strength in describing and 'following' both human and non-human actors, without privileging either (Callon 1986b, Latour 1987, Latour 1991, Latour 1999c, Law 1992) and it forms the other part of the theoretical framework. We take the main premise of actor network theory [ANT] to be that it considers it inappropriate to distinguish between the social and the technical world.

The function of this chapter is to explain and justify our use of ANT as the second part of our theoretical framework. Actor network theory is described briefly and then its use as constructivist, interpretivist and then realist is weighed. Some researchers urge the rejection of 'epistemological rivalry between realism and constructivism' on the basis that it is philosophically flawed (Tsoukas 2000). It is flawed because neither on its own is a fair expression of the 'truth'. There is an objective reality which is 'outside our heads' but at the same time we help create structures by our understanding of them which is institutionalized and 'historically adopted' (Tsoukas 2000).

Actor Network Theory

Actor network theory has evolved, as theories do, over the years since its conception, and its authors have diverged in their explanations of it. Actor Network Theory [ANT] was pioneered at the Ecole des Mines in Paris by Michel Callon (1986b) and Bruno Latour (1987). The very title of this theory has led to grave and perhaps misleading speculation (Latour 1999b, p5) as to its meaning. The authors admit that 'actor network' was perhaps not the most appropriate name for it but this was the closest translation from the French '*acteur reseau*' (Latour 1999b, p15). The effect that Callon and Latour desired was to emphasize a 'tension' between the two words actor and network, where you have the 'centred actor' and the 'decentred network' (Latour 1999b, p5). According to Law (1999) the misconception stems from the use of the word network in everyday language, especially with the rise of networked systems and the Internet. And this has caused further trouble with the concepts of space and time and how ANT deals with both. "The notion of 'network' is itself an alternative topological system. Thus in a network, elements retain their spatial integrity *by virtue of their position in a set of links or relations*" (Law 1999, p6). It is this relationship which is important as will be explained in more detail below.

One of the main and most widely discussed premises of actor network theory is its claim to consider the social and the technical world at the same level offering the notion of heterogeneity to describe projects and to offer a socio-technical account (Latour 1986, Law and Callon 1988). This avoids questions of: 'is it social?' or 'is it technical?', for this would be missing the point. ANT rather suggests that the question to ask in considering conspiring heterogeneous networks of intertwined social and technical elements is: 'is this association stronger or weaker than that one?' (Latour 1988b). This question captures well one of the main concerns of this work, which is to study OS as conspiring heterogeneous networks where there are machines, texts, developers, code, Internet infrastructure, users/databases and mailing lists etc, and within which we seek to find the "stronger" associations.

ANT is called a theory but its creators vacillate between both discussing and using it as a method or tool and a theory in the framework-providing sense (Callon 1999, Latour 2004a). In an amusing [insulting?] article where Latour is in a dialogue with an LSE student, Latour skilfully manages to draw out many fallacies held about actor network theory and how it is mis-applied to research. Latour disagrees with the word network and shifts the responsibility onto Callon for creating such an ambiguous name and then goes onto show how ANT doesn't say much about what we study but "about how to study things, or rather how not to study them. Or rather how to let the actors have some room to express themselves".

Latour (1987) suggests that ANT is based on three principles; agnosticism, symmetry of description and free association. Agnosticism means that the observer refrains from making any ethical or moral judgements concerning the protagonists in/under study. Symmetry of description emphasizes how social and technological issues are described using the same terminology or vocabulary [or each others][15]. Free association does away with the dualism between the social and the technical, thus neither of the two are privileged, in other words this is a 'repudiation of a priori distinctions between the social and the natural or technical' (Doolin and Lowe 2002) and such an approach 'starts from a closed definition of the social and then uses this repertoire as an explanation of nature' (Ormrod 1995) [see also (Callon 1986a, Callon and Latour 1992, Silva and Backhouse 1997)].

These principles have of course been sharply criticized (Collins and Yearley 1992a, Habers 1995, Pels 1995, Reed 1995, Winner 1993) in what has since been called the 'science wars' and even the 'Chicken debate' (Harbers and Koenis 1996)[16]. Collins and Yearley (1992a, 1992b) make a

[15] The point here is that though the human and technical do have a separate vocabulary a researcher should be able to use the vocabularies interchangeably when needed.
[16] http://www.chem.uva.nl/easst/easst961_2.html#harbers

number of criticisms but two of their main points will be considered here. They attacked Callon and Latour on their condition of symmetry between 'nature' and the 'social' and the idea that Callon and Latour are allowing or returning agency back to nature, "This backward step has happened as a consequence of the misconceived extension of symmetry that takes humans out of their pivotal role" (Collins and Yearley 1992a). They follow this with an assault on Callon and Latour's use of semiotics, implying that their approach does not help to understand the world in any significant way because 'discourse analysis has been largely abandoned within SSK' (Collins and Yearley 1992a, p305).

Winner's (1993) paper mainly criticized social constructivism but this was a critique of all the theories and approaches he believed were social constructivist in nature, including, but not only, ANT. His critique can be summarized in four main arguments. Firstly, he felt that social constructivists showed a 'total disregard for the social consequences of technical choice; secondly, it worried him that only certain voices were 'heard' in such studies, "who says what are the relevant social groups and social interests? What about groups that have no voice but that, nevertheless, will be affected by the results of technological change?; thirdly, he lamented the fact that this approach was narrow in the sense that it did not allow for the possibility that there may be 'dynamics evident in technological change beyond those revealed by studying the immediate needs, interests, problems, and solutions of specific groups and social actors' and; lastly, he sensed a reluctance, and disdain or even inability of such an approach to 'take evaluative stances or invoke moral or political principles' (Brey 1997, Winner 1993).

Many of these criticisms have been given a response by ANT's main proponents (Bijker 1993, Callon and Latour 1992, Latour 1991, Latour 1993, Latour 1996, Law 1991, Law and Bijker 1994, Star 1991) and/or were accepted and changes made to this theory by its pioneers to accommodate them. One of the key changes perceived is the more direct attitude taken by Latour (1999c) in his most recent work, *Pandora's Hope*, towards the criticism of being constructivist or realist, and this will be dealt with in more detail below. Callon and Latour (1992) responded to Collins and Yearley's (1992a) criticism of symmetry by making their point clear that the social and nature do indeed play significant roles and they are equal in the sense that there needs to be a certain degree of agency given back to non-human actants since without it we would not see the entire picture, "to pretend that to document the ways scientists bring in nonhumans, we sociologists should choose *one* of these positions - that scallops do not interfere at all in the debate among scientists striving to make scallops interfere in their debates - is not only counter-intuitive but empirically stifling" (Callon and Latour 1992). However, Latour and Callon make it clear that they don't 'wish to extend intentionality to things, or mechanism to humans', but just the ability to act, or as Rose and Jones (2004) describe it as "the capacity to make a difference".

Callon and Latour did not directly respond to Winner [indeed Winner never really attacked them directly either] but Mark Elam (1994) did instead, "Assuming high moral ground, he [Winner] accuses everyone linked with "social constructivism" of being, among other things, elitist, implicitly conservative, blasé, and politically naïve. Winner appears to be declaring war on something despicable..." (Elam 1994). He took Winner's last criticism of social constructivism's lack of moral stance and replied to that by pitting Woolgar's 'antirealism' against Winner's 'anti-constructivism'. He admitted that all he really wanted to achieve was to 'silence one without wishing to condone the other' (Elam 1994). This gave way to another battle of sorts with Winner responding (1994) likewise with more personal insults being aimed at one another rather than pure academic arguments, "he [Elam] seems more interested in applying ideological decals and passing peremptory judgments than in addressing any of the issues I posed. Presenting himself in the image of a "good liberal ironist," Elam shoots some rather illiberal, nonironic shots at "fear and loathing," "predilection for cruelty," and such like" (Winner 1994).

It is interesting to note that some of the criticisms made by Winner and Collins and Yearley highlighted, even then, the 'confused' position ANT adopted. It has taken Latour a number of revisions and improvements on clarity to make a clean admission to the middle position ANT holds between constructivism and realism. Collins and Yearley's comments about symmetry and giving agency to non-humans, especially, could be considered to have arisen from this ambiguity fostered by the authors of ANT. The more recent work of Latour (Latour 1999c, Latour 2005b, Latour and Weibel 2005) has made a strong attempt to refine the position of ANT.

ANT as Ontology

The use of ANT in this research is somewhat novel, for the intention here is to use ANT principally as an ontological position, while in other studies it is often subsumed under the socio-constructivist ontology of interpretivism. Indeed, ANT has proven to be innately flexible and it is this very flexibility which has allowed it to be 'folded' into interpretivist research, despite ANT's potentially holding a very decided ontology unique to itself (Cordella and Shaikh 2003). Most work using this theory such as Walsham (1993a), Hanseth and Monteiro (1996), Bowker and Star (1996) and Monetiro and Hanseth (1995a, 1995b), to mention a few, have focused explicitly on using ANT within an interpretivist and/or constructivist approach. Broadly stated, this ontology sees reality constructed in the interplay of actants[17] as they come together in some form of interaction; the interpretivist ontology argues that it is the interpreter who constructs reality in

[17] Actants is a term created by Latour and Callon to indicate both human and non-human actors, where actants are defined as "*Any element which bends space around itself, makes other elements dependent upon itself and translates their will into a language of its own*", (Callon and Latour 1981) (pp286).

his/her mind. Interpretivism sees reality as constructed in the acts of interpretation of a situation. In our reading of ANT reality comes into existence in the interaction, when more than one actants interact, and reality is not in the mind of any individual but is 'out there' in the interactions (Latour 1999a). In ANT the focus is on the constitutive forces found in the interplay among actors as they jointly define, constitute and construct this reality through associations (Latour 1999c, Law 1999). Such a theoretical stance is indeed critical towards constructivism, social-constructivism and hence interpretivism.

It follows that ANT does not only propose a new way of understanding reality; it also introduces its own epistemology, a distinct way of conceptualising the routes available to an understanding of that reality, what Latour (1999a) calls 'realistic realism' (Stalder 2000). This is explained in more depth later in this chapter under the section of 'operationalizing the ontology of ANT', but suffice it to say here that Latour has steered away from 'conventional' constructivism to create a more flexible approach to the research of socio-technical hybrids (Lee and Hassard 1999).

The argument is that ANT has its own distinct ontology which is "more objective than objectivists" (Latour 2001), and thus there is little room for interpretivism even if ANT has often been forced to adopt the ontology of interpretivism and thus suppress its own ontology. Interpretivism is certainly socio-constructivist in nature, but ANT is not. The ontology of interpretivism deems the interpreter to be in a position to construct reality in his/her mind whereas, ANT accentuates that reality is constructed by the interplay of two or more actors (both human and non human), and this reality emerges from such interaction. Therefore for interpretivism, reality is created in the mind, while for ANT, reality emerges 'out there' as action and interplay – it is performed. These differences in the ontological foundation of ANT and interpretivism should be taken into consideration when ANT is used to inform research in IS. These differences impact the very defining of the object of analysis and thus also the outcome. However, before ANT's ontology is clarified in more detail there is a need to go back to interpretivism and constructivism.

Interpretivism

Interpretive research can be distinguished from other kinds of research by the underlying philosophical assumptions which guide the work (Klein and Myers 1999, Myers 1997). Interpretive studies generally attempt to understand IS phenomena through the meanings that people assign to them, and interpretive methods of research in IS are 'aimed at producing an understanding of the context of the information system, and the process whereby the information system influences and is influenced by the context' (Garcia and Quek 1997, Walsham 1993b, Walsham 2006). Fitzgerald and Howcroft (1998) claim that in interpretivism there is no universal truth, uncommitted neutrality is impossible and realism of context is important. An alternative paradigm is suggested by them, critical theory, which is 'characterized by a realist ontology allied to a subjectivist epistemology' (Fitzgerald and Howcroft 1998).

If reality cannot make direct reference to the world "itself," but needs some intermediation so that nature and social reality are constructed *via* this intermediation, the task of the researcher is to interpret and hence explain the processes that are "producing" the phenomenon. Interpretivism, like positivism, is itself socially constructed, populated with social science researchers whose shared beliefs include the four concepts outlined by Lee *et al.* (1997). First, the subject matter of interpretative research involves the 'life world', which includes humanly created meanings, be they individually held or those shared by groups. Second, the researcher himself must inevitably serve as an instrument of observation. Third, interpretation is iterative (hermeneutic) and lastly the validity of an interpretation can be assessed (Lee *et al.* 1997).

The ontology [nature of reality] would be where each person is considered to construct his or her own reality (Walsham 1995a, Walsham 1995b). The epistemology [nature of knowledge claims] of interpretivism is that 'facts and values are intertwined and hard to disentangle, and both are involved in scientific knowledge'. We all have our own 'realities' as Walsham states, but the importance of this approach is in understanding how every person's reality connects with another persons. For interpretive researchers it is when people create 'shared realities' that it becomes interesting to study. Thus in most interpretative approaches, a central idea is 'mutual understanding' – the phenomenon of a person understanding (i.e. 'interpreting') what another person means, whether it is a person engaged in everyday life taking a *natural attitude* to understanding another person in everyday life, or it is a person engaged in scientific research taking a calculated *scientific attitude* to understanding everyday people/phenomena in their everyday lives (Lee *et al.* 1997).

Much interpretive information systems research is characterized by an explicit intention to understand the relation of information technology to organizational activity through 'an understanding of the *context* of the information system, and the *process* whereby the information system influences and is influenced by its context' (Walsham 1993b). This is based on the belief that: 'the same physical artefact, the same institution, or the same human action, can have different meanings for different human subjects, as well as for the observing social scientist' (Lee 1991). Within such IS research interpretivist approaches acknowledge that, although information systems have a physical component which permits their technical operation, they are designed and used by people operating in a complex social context (Doolin 1998). Thus, an information system is understood (constructed) differently by different individuals, and is given meaning by the negotiated shared understanding of such phenomena which arises out of social interaction: "Events, persons, objects are indeed tangible entities. The meanings and wholeness derived from or ascribed to these tangible phenomena in order to make sense of them, organize them, or recognize a belief system, however, are *constructed realities*" (Lincoln and Guba 1985). Much interpretivist research has also been constructivist in nature, though the two are by no means inseparable. ANT in particular has been labelled as constructivist, and till rather recently the authors of this theory themselves admitted to the same.

Constructivism and Social Constructivism

Crotty (1998) endeavours to define constructivism, constructionism and then social constructivism. The first two are very close but differ in the respect of individual versus a collective construction of reality. Crotty explains that constructionism is not about creating meaning but about *constructing* meaning. *Constructivism*, according to Crotty, is a term "reserved for epistemological considerations focusing exclusively on 'the meaning-making activity of the individual mind' and proposes to use *constructionism* where the focus includes 'the collective generation [and transmission] of meaning'" (1998, p58). Social constructionism or constructivism mean different things to different people. Crotty explains the difference effectively when he says that some "understand social constructionism as denoting 'the construction of social reality' rather than 'the social construction of reality'. The difference lies in whether you believe that there is an objective reality 'out there' which you then understand in your own way [which is the social construction of reality] or if you believe that all reality is socially constructed and no object or thing has any significance other than the one given to it; a chair is a chair only because we 'see' it to be a chair and if we hadn't been told that it is a chair we may not even have noticed it at all.

Latour (2003) also makes a distinction between constructivism and social constructivism, as does Crotty, but many writers in this field conflate these concepts and thus use these terms interchangeably. This is the reason why Sismondo (1993) and others who have tried to provide a taxonomy of constructivism end up by suggesting that it is not worthwhile to make a distinction. Perhaps this could help to explain somewhat how the 'science wars' went on for as long as they did because there are and were too many diverging definitions of even constructivism, let alone actor network theory. Latour (2003) believes such a distinction is necessary because the word 'social' before constructivism gives the wrong impression that 'the construction is made of social stuff' rather as if it implied certain ingredients and that is why he insists on using 'constructivism' and more recently, constructive realism (Latour 1999c).

Writers such as Bijker, Hughes and Pinch (1987) in the 1980's could be called the pioneers of the social constructivist approach (Brey 1997). Not surprisingly, this approach is frequently considered synonymous to the Social Construction of Technology [SCOT] approach of Pinch and Bijker (1987). Berger and Luckmann (1966) may not have invented the phrase 'social construction' (Sismondo 1993), and albeit the recent understanding of this phrase is quite diverse from theirs (Knorr Cetina 1993), but their book *The Social Construction of Reality* laid the foundation for sociology studies and merits a mention here. This book has two parts, one which is devoted to 'society as objective reality' [the 'out there' position] and the other to 'society as subjective reality'. However, before they begin on these they spend some time discussing two social processes, legitimation and institutionalisation. They then explain subjective and objective reality in terms of the above social processes. Society as objective reality is explained through an example of a child moving to a new place and, as everything is new, how he accepts many rules as they are, so for him this is an objective reality. In other words it already exists out there and he begins to accept the rules and culture of the place. However, as soon as you begin to follow the rules of a society, even after coming to a better understanding of them, and perhaps not even agreeing with them you still reify them because as everybody else takes them to be 'real' you end up having to reinforce them as well. This locks in both social processes of legitimation and institutionalisation and also explains society as subjective reality. So we move from an objective reality to a subjective one through the process of first legitimising rules, which then over time institutionalises them, and as such everybody contributes to their social construction.

Orlikowski and Baroudi (1991) continue with the tradition to define constructionism and social constructivism. They attempt to distinguish, perhaps not too convincingly, between weak and strong constructionism, where weak constructionism denotes minimal or no intervention of the researcher into the study, thus it is more about describing the agents and what and why they are doing what they do. However, 'strong' constructionism is described more as an intervention by the researcher because in order to make sense of the study s/he needs to be a part of it and thus construct it too. The former may be allowed by positivists but the latter is the exact opposite. Though a distinction is made between the two types it is less than convincing. Brey (1997) develops another classification, based roughly on the categories developed by Bijker and Law (1992), Sismondo (1993), Collins and Yearley (1992a), Woolgar (1991) and Grint and Woolgar (1995). His taxonomy included *strong* social constructivism, *mild* social constructivism and *actor network theory*.

Strong social constructivism, according to Brey (1997), is affected by a need to 'vigorously uphold the principle of symmetry' where 'technology is a genuine social construction' and genuine here implies stabilization. This includes the SCOT approach. *Mild* social constructivism refers to the 'more moderate' approaches such as the 'social shaping' ones developed by MacKenzie and Wajcman (1985) and MacKenzie (1990). This approach differentiates between society and technology and though they admit that technologies do have a role and even properties they come back to the belief that this is only so because technology is socially constructed. Actor network theory (Callon 1987, Callon and Latour 1992, Latour 1987) is similar to strong social constructivism but the difference is that ANT relates to a generalized symmetry where the emphasis is not on the social elements, indeed, stabilization is 'not the result only of social factors' (Brey 1997).

Hacking (1999, p6) came up with yet another classification which is summarized in his words, "Social constructionists about X tend to hold that: (1) X need not have existed, or need not be at all as it is. X, or X as it is as present, is not determined by the nature of things; it is not inevitable. Very often, he suggests, they go further, and urge that: (2) X is quite as bad as it is. (3) We would be much better off if X were done away with, or at least radically transformed". Luckily we do come out of that tangle with Hacking (1999, p48) supplying us with a definition of social constructionism, "Hence by *constructionism* (or social constructionism if we need, on occasion, to emphasize the social) I shall mean various sociological, historical, and philosophical projects that aim at displaying or analyzing actual, historically situated, social interactions or causal routes that led to, or were involved in, the coming into being or establishing of some present entity or fact".

Sismondo's (1993, p516) constructivism classification consists of "(a) the construction, through the interplay of actors, of institutions, including knowledge, methodologies, fields, habits, and regulative ideals; (b) the construction by scientists of theories and accounts, in the sense that these are structures that rest upon bases of data and observations; (c) the construction, through material intervention, of artefacts in the laboratory; and (d) the construction, in the neo-Kantian sense, of the objects of thought and representation". But what interests us is Sismondo's (1996) 'creeping realism'. This he equates with what he calls Latour's 'heterogeneous constructivism' because Latour's ideas involve both human and non-human but also because such a take on constructivism is prone to 'realism'. So where does ANT stand then, within constructivist approaches or realist?

Part of that question can be answered by retracing the difference between constructivism and constructionism. If one does try to categorize ANT as either constructivist/constructionist or objectivist, it needs to be clarified again that constructivism is closer to objectivism than constructionism could ever be. "Constructivism describes the individual human subject engaging with objects in the world and making sense of them. Constructionism, to the contrary, denies that this is what happens... Instead, each of us is introduced directly to a whole world of meaning" (Crotty 1998, p79). As a result, constructionism, by making it clear that there is a reinforcing culture already present which is constructed socially leaves little room for objectivist ideas [ideas such that 'truth and meaning reside in objects independently of any consciousness' (Crotty 1998, p42)]. Again, Orlikowski and Baroudi's (1991) weak constructionism, because this allows a certain amount of freedom to the objects of study to 'speak' for themselves, does let ANT squeeze through onto a common ground, albeit a rather grey one.

Numerous taxonomies have thus been developed to make sense of the diverse work carried out under the umbrella term, constructivism. The significance of such taxonomies for this work lies in the question as to where, or if even, ANT can be placed in the work called constructivist. As can be seen from the above classifications, there is no one definitive definition of constructivism, and there are far too many differing classifications.

Ontology of ANT

"All the absurdities [which] I have disputed for twenty-five years [are]: that science is socially constructed; [and] that there is no reality out there; ... such nonsense"(Latour 1999c, pp299-300).

We are meant to be surprised when Latour's friend asks him the question "Do you believe in reality?" (Latour 1999c, p1) but there is more to that cryptic question than is apparent. Where is this reality, and where do we study it? Is it 'out there' or in our minds? "Why, in the first place, did we even need the idea of an *outside world* looked at through a gaze from the very uncomfortable observation post of a mind-in-a-vat?" (Latour 1999c, p12) Latour stresses that we don't need to construct an outside world, just so that we can sit and view it from the 'inside' because there is a reality but it is not 'out there'. Reality is created in the very act of actants coming together in some form of interaction, "There is no sense in which humans may be said to exist as humans without entering into commerce with what authorizes and enables them to exist (that is, to act)" (Latour 1999c, p192) and Latour repeats the same for non-humans. Callon (1987) explains the same concept of ANT, but from the point of view of the network, "The actor network is reducible neither to an actor alone nor to a network.. An actor network is simultaneously an actor whose activity is networking heterogeneous elements and a network that is able to redefine and transform what it is made of".

There is thus a strong objective stance within ANT, and Latour (2005b) makes this claim more openly and strongly in his recent article than in any of his previous work, even Pandora's Hope, so when the imaginary LSE student asks him, "Surely you don't want to say you are of the objectivist type?" Latour replies with an emphatic "You bet I am!". Of course this confession is not without some problems. Isn't the idea that we are interpretivist and subjective human beings taken as a given? Latour's response to any such claim is that when we are enjoying the sight of some beautiful sculpture it is not us, the viewer, who brings subjectivity to the situation but the object instead because, "*If you can have many viewpoints on a statue it's because the statue itself is in three-dimensions and allows you, yes, allows you to turn around it. If something supports many viewpoints, it's just that it's highly complex, intricately folded, nicely organized, and beautiful, yes, objectively beautiful*".

Actor network theory, seen in this way, clearly sets itself against the notion of constructivism. ANT maintains that the constitutive forces in the interplay among actors themselves define, constitute and construct this interplay (Latour 1999a, Law 1999). This theoretical stance is indeed critical towards constructivism, social-constructivism and hence interpretivism. The *essence* of the theory, as used here, stands in the argument of the co-definition and co-evolution of objects and humans, both called indistinctly, actors. Thus, the constitutive essence of actor network theory cannot be confused with the constructivist assumption of interpretivism. It follows that ANT does not only propose a new way of questioning reality, it also introduces a new way of conceptualising the understanding of reality. What Latour (1999c) calls 'realistic realism' (Stalder 2000). This argues that reality does not exist *per se,* but states that the construction of reality is achieved through the interplay between different actors, both human and non-human, with symmetric constitutive characteristics (Latour 1987, Law 1992, Law 1999).

It is not altogether true that ANT is constructivist but then again nor is it convincing to say that ANT is not constructivist. It may be better understood as a particular flavour of constructivism which is not conventional at all but does involve certain elements, or how Sismondo (1996) would describe 'creeping realism'. In ANT information technology and users are not defined outside their relationship but *in* their relational networks. This consideration moves the focus of the analysis from the actor, either human or non-human, towards a more complex and less defined phenomenon, that is the interaction. This change in focus not only affects the analysis of the phenomena, but also the assumptions about the nature of the entities that constitute the phenomena. Actor-network[18] theory rather incites us to reconsider sociotechnical relationships as an open ended set of interactions where the actors of the sociotechnical interplays do not pre-exist the relationships; the actor is generated *in* and *by* these relationships. It has a "relational materiality" (Law 1999) i.e. actors achieve their form and attributes as a consequence of their

[18] Originally this theory was named with a hyphen between the words actor and network but most of the time this hyphen is either dropped or ignored, even by Latour and Callon. Latour (Latour 1999b) (pp15 and 21) made an explicit mention of this hyphen to explain why this is one more reason why ANT has been misunderstood. It draws one back into a discussion over agency and structure which according to Latour actually emerges from a deeper problem stemming from a 'world outside' and a subjective inside.

relations with other actors. This reflects an aversion to accept *a priori* the pre-existence of social structures and differences as somehow intrinsically given in the order of things. There are no distinctions between social and technical subsystems. It is the relationship that produces the actors, emergent from the very interplay among different, human and non-human, entities. There is no case of a local or global dimension here, only a relational one (Latour 1999a).

However, and this is where the complexity and power of the theory emerges, the actors that are part of the network, where Callon defines a network as a "group of unspecified relationships among entities of which the nature itself is undetermined" (Callon 1993), are also the constitutive elements of the network. Therefore, the concept of actor and network are concatenated and cannot be defined without the other – thus we have an actor-network. "The actor network is reducible neither to an actor alone nor to a network....An actor network is simultaneously an actor whose activity is networking heterogeneous elements and a network that is able to redefine and transform what it is made of" (Callon 1987). In ANT actors are not defined and analysed in a stable set of relationships. It is the researchers who artificially define the analytical range of the study to see "what the various actors in a setting are doing to one another" (Akrich and Latour 1992). By limiting the level or focus of the investigation it is possible to study and understand some of the relationships that are shaping both actors and the relational networks. However, it must remain clear that actors and actor-networks are naturally *embedded* in open ranges of relationships that cannot be artificially limited by the scope of any particular analysis. Actor network are "open ended" and can only be artificially (but usefully) closed and isolated from the broad and natural openness of relationships.

The complexity of the relational pattern in an actor-network is given shape and structure when all the actors in the actor-network embody the same level of flexibility. Actors embody various characteristics that are the outcome of their relationships with "heterogeneous elements, animate and inanimate, that have been linked to one another for a certain period of time" (Callon 1987). These characteristics are renegotiated in the interplay with other actors. An actor-network embodies these characteristics so that the outcome is the result of "a set of diverse forces" (Akrich 1992) that affect and define the inter-networked relational settings. These forces can be considered as an embodiment of prescriptions by the actors. A prescription is defined as "what a device allows or forbids from the actors -humans and non-humans - that it anticipates; it is the morality of a setting both negative (what it prescribes) and positive (what it permits)" (Akrich and Latour 1992). A point worth mentioning here is that as can be seen networks do have their own morality; but ANT doesn't want the researcher to bring his/her morality to the study because this would then colour what the 'reality' is as reality is created by the actors being investigated. This responds to Winner's fourth criticism about how constructivism, and particularly ANT, fails to take a moral stance.

If an actor is analysed in isolation (taken out from the network) it may be seen to embody specific inscribed characteristics that may strongly affect the configuration of the contextual relational network under analysis. But people and technology "are never located in bodies and bodies alone, but rather that an actor is a patterned network of heterogeneous relations, or an effect produced by such a network" (Law 1992). As a consequence, actors do not embody, in isolation, action or *actantiality* (potential for action) but it is their relational dimension that generates instances of action (Latour 1999a, Law 1992). Such actantiality is generated in a process of negotiation. A process which is circular and recursive, the course that defines and redefines actors in their multiple contexts. Actors are in action and, as a consequence, in a continuous state of mutation. This continuous relational interplay is the *performative* characteristics of actor networks, where actors are in fact "*performed* in, by, and through relations" (Law 1999, p7).

Actors, in their interplay within the actor-network, negotiate their forces in a process of translation. "By translation we understand all the negotiations, intrigues, calculations, acts of persuasion and violence thanks to which an actor or force takes, or causes to be conferred to itself, authority to speak or act on behalf of another actor or force." (Callon and Latour 1981). Translation is the circular process of "interpretation" or as Callon (1991) puts it, the "definition" that every actor makes of other actors in the actor-network.

Finally, recalling the concept of circularity of the actor-networks relations, it is clear that every actor-network affects and is affected by the characteristics of the actors and then by the different interests the actors bring to the actor-network. Every actor can bring characteristics that have emerged from other actor networks to which it belongs, because an actor can and usually does belong to more than one actor-network at one time. Recursively, new, emergent characteristics are re-proposed back into the other actor-networks. This circularity can explain the action that is endogenous to the relational interplay analysed by actor-network theory.

Constructive Realism or Realistic Realism[19]?

Lee and Hassard (1999) suggest that ANT is '*ontologically relativist*, in allowing the world to be organized differentially, yet *empirically realist* in providing "theory-laden" descriptions of organization'. Such 'ontological slipperiness', stress Lee and Hassard (1999) and even Doolin and Lowe (2002), can be very valuable in studying organizations, and their practices and forms,

[19] Latour's phrases from Pandora's Hope (1999) p135 and p15, respectively.

because "if we are to have successful organizational form *now*, it must emerge through interaction and negotiation, and if we are to respect organizational form as an emergent property of relationships, we must allow that form to change at a moment's notice" (Lee and Hassard 1999). The authors explain how this ontological relativity and empirical realism makes ANT more 'flexible'. This is based on their idea that ANT never created boundaries, indeed Latour and Callon made a real attempt to keep this theory in a 'state of permanent revolution' (Lee and Hassard 1999). As Latour (1999b, p20) points out, "From the very beginning, ANT has been sliding in a sort of race to overcome its limits and to drop from the list of its methodological terms any which would make it impossible for new actors (actants in fact) to define the world in their own terms, using their own dimensions and touchstones".

Latour sums up his dilemma with ANT by drawing on the example of Pasteur at work on his lactic acid yeast in his laboratory "An experiment *shifts out*[20] action from one frame of reference to another. Who is acting in this experiment? Pasteur *and* his yeast. More exactly, Pasteur acts *so that* the yeast acts alone. We understand why it is difficult for Pasteur to choose between a constructivist epistemology and a realist one: he creates a scene in which he does not have to create anything. He develops gestures, glassware, protocols, so that the entity, once shifted out, becomes automatic and autonomous. According to the ways in which these two contradictory features are stressed, the same text becomes constructivist or realist" (Latour 1992). This is the crux of what Latour calls constructive realism.

So, for Latour (1999c, p296) "There is no world outside, not because there is no world at all, but because there is no mind inside, no prisoner of language with nothing to rely on but the narrow pathways of logic." He (Latour 1999c, p281) goes on to explain what he means by constructive realism through an analogy of creation,

> "...whenever we make something *we* are not in command, we are slightly *overtaken* by the action... Thus the paradox of constructivism is that it uses a vocabulary of *mastery* that no architect, mason, city planner, or carpenter would ever use. Are we fooled by what we do? Are we controlled, possessed, alienated? No, not always, not quite. That which slightly overtakes us is *also*, because of our agency, because of the *clinamen*[21] of our action, slightly overtaken, modified. Am I simply restating the dialectic? No, there is no object, no subject, no contradiction, no *Aufhebung*[22], no mastery, no recapitulation, no spirit, no alienation. But there are events. I never *act*; I

[20] '*Shifting out*' is a semiotic expression that describes the possibility of displacing action into another time, another space, or another actant. On its use in scientific text, see Latour (Latour 1988c)
'*Shifting in*' means closing the first shift by reverting to the original frame. I have introduced "shifting down" to express the delegation to technical objects: see Latour (Latour 1994).

[21] An inclination, bias - The round and the square would, by certain clinamina, unite. An insensible clinamen (to borrow a Lucretian word) prepares the way for it. No old word; which, with a slight clinamen given to its meaning, will answer the purpose.

[22] Abolition

am always slightly surprised by what I do. That which acts through me is also surprised by what I do, by the chance to mutate, to change, and to bifurcate."

Constructive realism is then a way of expressing how agency is passed to non-human actors. Thus allowing them to really make a difference instead of just passive objects. Non-humans can in turn pass back some of this agency to humans again. Latour and Johnson (Latour 1988a, Latour and Johnson 1995, p263) calls this prescription, "the behaviour imposed back onto the humans by nonhuman delegates. Latour's ideas of constructive realism are operationalized below and Latour's constructive realism can be seen 'in action'. Suchman (1987, 1998) links the idea of agency with performativity and stresses that "the price of recognizing the agency of artifacts need not be the denial of our own. Agency – and associated accountabilities – reside neither in us or nor in our artifacts, but in our intra-actions. The question ... is how to figure our differences in such a way that we can intra-act responsibly and productively with and through them".

Operationalizing the Ontological Concepts of ANT

In order to apply the ideas of constructive realism we must operationalize them. Latour and Callon have provided us with such vehicles in the form of circulating references etc but a very simple way to understand this would be, "If we display a socio-technical network – defining trajectories by actants' association and substitution, defining actants by all the trajectories in which they enter, by following translations and, finally, by varying the observer's point of view – we have no need to look for any additional causes. The explanation emerges once the description is saturated" (Latour 1991, p129). Simply, but exquisitely put, this describes one way ANT can be operationalized because ANT is about intense description which leads the researcher to understand and 'see' what transpired and how reality was created. The researcher must allow the actants to 'speak'[23] for themselves and not put words in their mouths. Any attempt to interpret the actants in the heterogeneous networks would invalidate the first principle given by Latour, and Amsterdamska (1990) would seem to agree. Moreover, using the ontological dimension of ANT, this attempt would simply become an action of merely *one* of the actors in the network instead of an interplay, thus it seems that ANT considers reality to be "emerging out there" while interpretivism states that reality is constructed via interpretation.

We can also look to certain tools which might be used for this explanation and description; circulating references, immutable mobiles and the concept of centres of calculation. This research has used a combination of these three, along with ideas of shifting in and out to explain how 'learning' circulates, grows, changes and evolves in open source software creation.

[23] For more on actants 'speaking' see Pouloudi, A. and Whitley, E. A. (Pouloudi and Whitley 2000)

Circulating Reference

The concept of circulating reference is used both as an instrument which allows something to be studied and at the same time it clarifies what Latour means by constructive realism. In order to study something much too large or complex, like the forest in Pandora's Hope or a large community of open source developers, you need to be able to extract from that complexity something manageable and transportable. However, you then risk losing any sense of the greater object. Latour believes that scientists rely on what he calls the circulating reference to keep the link between the samples they collect and the more complex whole that those samples were a part of. These samples are a representation of the original object of study.

The origin of the word reference is the Latin '*referre*' meaning "to bring back" (Latour 1999c, p32). Latour's use of this word is subtle because the 'reference' both extracts from the object of study but then draws this information back into the line of discourse. There is a heavy leaning into semiotics and the concept of circulating references draws on this area through its use of 'internal referents' and both 'shifting in' and 'shifting out'. A reference is a sign which signifies something else, thus the sign is an indication for something. Latour gives the example of the forest of Boa Vista and how the soil samples taken by the scientists are a reference of the larger forest. They are a part of the forest but in the laboratory of scientists these references play the role of 'being' the forest. Now how is this leap between the object of study to its reference actually achieved? A laboratory dish full of a soil sample does not immediately conjure up the image of a real forest and yet to the scientists this soil is a way not only of 'seeing' the forest but of magnifying it.

How is a link made between the object and the reference in the first place? Many things, words or even ideas can be a reference for an object because "one never travels directly from objects to words, from the referent to the sign, but always through a risky intermediary pathway" (Latour 1999c, p40) . In Latour's study of the scientists at work in the forest it was pictures of the forest, branches from trees, soil samples, labels on various trees, maps etc. In this research on open source a single thread of conversation on a specific topic printed out is a reference for the larger community of open source developers. However, going back to the forest, a branch snapped off and taken back to a laboratory is a reference, not only for that tree but perhaps the entire forest as well. The branch will be carefully labelled, coded, classified, categorized and stored in a specific area. It is this process that this branch goes through from selection onwards to being studied and stored in a laboratory that helps to create the reference, and "what a transformation, what a movement, what a deformation, what an invention, what a discovery! In jumping from the [tree] to the drawer, the [tree branch] benefits from a means of transportation that no longer transforms it. Having made the passage from a [tree branch] to a sign, the [branch] is now able to travel through space without further alterations and to remain intact through time" (Latour 1999c, p51).

A strong connection between the object and the reference is essential, and this 'gap' is traversed by a series of tiny connections "we never detect the rupture between things and signs... We see only an unbroken series of well-nested elements, each of which plays the role of sign for the previous one and of the thing for the succeeding one" (Latour 1999c, p56) . Each tiny connection connects to the previous connection but not to the one before that so in effect we have a chain where if one link is broken then the reference fails to make the link to the object so, "No step – except one - resembles the one that precedes it, yet in the end, when I read the field report, I am indeed holding in my hands the forest of Boa Vista. A text truly speaks of the world. How can resemblance result from this rarely described series of exotic and miniscule transformations obsessively nested into one another so as to keep something constant?" (Latour 1999c, p61). However, one thing that needs to be pointed out here is that this is about *aligning* these connections and not about resembling the preceding ones.

Now we have the link between the object and reference but what purpose does this serve? This brings us back to how these references serve as magnifying devices. Latour explains this through 'reduction' and 'amplification'. The branch reference no longer shows us the finer details available to us if we study the real and entire forest and we also 'lose locality, particularity, materiality, multiplicity, and continuity', but on the other hand this reference now allows us to gain 'greater compatibility, standardization, text, calculation, circulation, and relative universality' (Latour 1999c, p70). These changes are respectively called reduction and amplification.

For Latour (Latour 1999c, p80) facts circulate and are followed in order to reconstruct the system, "Phenomena are what *circulates* all along the reversible chain of transformations, at each step losing some properties to gain others that render them compatible with already-established centres of calculation" (Latour 1999c, p71-72). Thus the circulating reference is called thus because it is always in motion and while in motion it tends to take on new layers and facts while dropping others, in other words it is produced 'when immutable mobiles are cleverly aligned' (Latour 1999c, p307). This is a simple but persuasive idea which is evident in open source development. Patches move in search of acceptance, and on this journey they both lose and gain properties as they pass from actor to actor.

How does the circulating reference concept help explain ANT as realist and objectivist? The circulating reference is 'constructed' *by* the scientists in steps but eventually becomes *more* realist because it speaks to us louder and clearer then could the real object. For example, the Boa Vista forest was too large to study and understand standing in it, but by collecting the samples and reconstructing the forest in a laboratory, the scientists were able to make real sense out of it. From this point on the references took on a role of their own and began to interact and construct reality in their acts of interaction. So a reference "does *more* than resemble. It takes *the place of the original situation*, which we can retrace, thanks to the [references]". However, 'it replaces without replacing anything. It is a strange transversal object, an alignment operator, truthful only on condition that it allow for *passage* between what precedes and what follows it" (Latour 1999c, p67). And of course here "the stress is on the granting of agency – the inscriptions have to be inscribed and the immutable mobiles have to be created and to gain assent. But once the agency has been granted, the inscriptions gain a degree of autonomy. They can compel further assent. It is in this that their immutability lies" (Collins and Yearley 1992a) and the 'realism' of Latour's constructive realism.

Amsterdamska (1990), in a review of Latour's *Science in Action*, criticized ANT and Latour for trying to strike an uneasy balance between his realism and constructivism. This critique hinged on Latour's first principle, which states, "The fate of facts and machines is in later users' hands; their qualities are thus a consequence, not a cause, of collective action". Amsterdamska (1990) insists that this is the most direct proof of the social constructivist nature of ANT. However, I disagree with her interpretation because though I understand that in his early work Latour never explicitly made any claim here he aims to show that it takes more than one inscription to build a circulating reference. The point Latour is trying to make in the first half of the sentence is that fate of any fact is left to the people who follow this experiment over and over in order to bring out some new element each time. Knowledge here is accumulative and dependent on how strongly the immutable mobiles are aligned to stop the circulating reference from losing momentum. It is not the first person who carries out this experiment who is the most important, it is all the people who follow it up and test it because this is the process of creating a circulating reference. How these people choose their reference and then allow *it* to amplify itself is important and that is why this is a consequence and not a cause of collective action. The circulating reference takes on a role and life of its own and thus constructs its own reality.

Centres of Calculation and Immutable Mobiles

Closely linked to the concept of circulating reference is that of centres of calculation. Immutable mobiles are created and combined in 'centres of calculation', and immutable mobiles allow centres of calculation to 'act at a distance'. Centres of calculation are concerned with making implicit knowledge explicit. The process of knowledge acquisition takes a scientist/ researcher through the steps of gathering *local* knowledge from actants and converting it to *universal* knowledge which then helps to shape the 'fuzzy beliefs' gathered from the actants into 'certain and justified *knowledge*' (Latour 1987, p216). Knowledge, here means, "how to bring things back to a place for someone to see it for the first time so that others might be sent again to bring other things back. How to be familiar with things, people and events, which are *distant*" (Latour 1987, p220). Latour uses the example of early explorers venturing forth to discover new lands. The explorers, through their interaction with the local people, gathered knowledge which they brought back with them. The first explorer didn't have any past collected knowledge to guide him in this discovery but when he takes the knowledge he gains back with him and renders it more logical and compact, he then has some legacy for the next group of explorers who follow in his footsteps. If this next group have had access to the first explorer's collection of knowledge then they won't see the new land for the first time even for themselves, because the *first* 'glimpse' of the land will have come to them through their perusal of the collected knowledge. So now when they do go to this land they will be in a position to build on the knowledge gained by the previous explorers and bring this new knowledge back to add to the knowledge base.

Thus centres of calculation act at a distance, which means that they somehow bring 'home' events, 'places and people' which are in fact distant and they manage this by 'inventing' ways to make them mobile, stable and combinable, "a) render them *mobile* so that they can be brought back; b) keep them *stable* so that they can be moved back and forth without additional distortion, corruption or decay, and c) are *combinable* so that whatever stuff they are made of, they can be cumulated, aggregated, or shuffled like a pack of cards" (Latour 1987, p223). This is the process of 'calculation' and at the centre is the events, people and places which are being changed into immutable mobiles.

These centres of calculation or traces also make backtracking possible, so one can move from the centre to the periphery and back again. And it is this calculation and backtracking that, once caught in the centre, can then be condensed onto 'paper'. Like circulating references, or perhaps because of the link between them and centres, the traces in the centres have to be tightly linked together. Each link leads to a greater degree of abstraction, "which is the process by which each stage extracts elements out of the stage below so as to gather in one place as many resources as possible" (Latour 1987, p241). Statistics, according to Latour is a fine example of abstraction, managing very succinctly, to condense a large amount of data, but this data is linked closely to the phenomenon of study and the scientist can trace his way back from this centre to the object of study.

Moser and Law (2006) differentiate between calculation and qualculation. They understand immutable mobiles to be mutable rather than immutable. Calculation, they insist, can be 'taken to be mechanical and algorithmic in form. The implication is that it demands information that is relatively clear and codified, information that holds its shape' (2006, p58). However, they argue that often information does not hold its shape, and instead it changes as it moves so rather than being immutable it is a mutable mobile. As information circulates it changes and adapts both its content and form, thus information is performative and able to generate something novel. This nature and ability of information they name as qualculability. It is through performance that agency emerges, a hybrid agency that belongs to neither human or non-human exclusively but only comes into existence as part of the performance of the collectif (Callon and Law 1995), thus 'politics is no longer limited to humans'(Latour 2005a, p31). This multifaceted and interesting idea of a mutable mobile is something we intend to explore in more depth later in the thesis.

Centres of calculation and circulating references are also intertwined because it is the centres of calculation that enable creation of immutable mobiles which can go on to be strongly aligned to become a circulating reference. And both these concepts are extremely useful for this research because when used together they will help to analyse and explain how learning occurs in open source communities and more specifically through version control tools. These ANT concepts allow for the detail that reality is actually constructed in the very interaction of actants and is not a construction in a person's mind. There is a "totally different 'outside' now that epistemology has been turned into a circulating reference" (Latour 1999b, p23).

Version control tools are a way of "transforming the juxtaposed set of allies into a whole that acts as one to tie the assembled forces *to one another*, that is, to build a **machine**. A machine, as its name implies, is first of all, a machination, a stratagem, a kind of cunning, where borrowed forces keep one another in check so that none can fly apart from the group" (Latour 1987, pp128-129). Open source is a strange blend of technology and humans where you have developers, code, tools, the Internet, different communication and governance modes. Thus it lends itself to an ANT scrutiny. Kavanagh and Araujo (1995) explain that "within [a] script are inscribed a series of instructions called a programme of action (Akrich 1992). A simple example of a programme of action is a line of code or 'an IF ...THEN statement' (Kavanagh and Araujo 1995). The writing of a programme of action is a process of inscription and when these programmes of action are successfully enacted they create a transformation and thus time (Kavanagh and Araujo 1995). If one were to make a fleeting or superficial analysis of open source one could say that it is all about creating code of which the source can be read, changed, amended and learnt from. And the above example from Kavanagh and Araujo (1995) shows quite neatly how ANT aids in doing this and more.

Circulating references and centres of calculation take us further than just a slim description. They help to explain how learning itself happens in open source communities. Both concepts show how a strong and unbreakable link is needed to allow for following the traces but more importantly they reflect how learning is about small additions being made to something over time which allow every actant that comes in contact with it to grow and expand in one respect but also the ability to funnel and sift out the key features needed, in other words abstraction. It is all about adding to some findings and getting a better 'view' because you are now sitting on a greater number of inscriptions and can thus 'see' further and have a better understanding, "it is important for us to do justice to the cleverness of this additional work going on in the centres without exaggerating it and without forgetting that it is just that: *additional* work" (Latour 1987, p236).

Every version of software is a combination of knowledge gained and accumulated slowly as it moves further and further away from the original so "If we fail to recognize how much the use of a technique, however simple, has displaced, translated, modified, or inflected the initial intention, it is simply because we have *changed the end in changing the means*, and because, through a slipping of the will, we have begun to wish something quite else from what we at first desired. If you want to keep your intentions straight, your plans inflexible, your programmes of action rigid, then do not pass through any form of technological life. The detour will translate, will betray, your most imperious desires" (Latour 2002, p252). If we consider what constructive realism *can* help us to see and explain then perhaps it will be easier to understand why a pure constructivist approach would not have been useful or perhaps even half as interesting. In open source learning and software development go hand in hand and this is what creates reality. Actors in the form of developers *allow* the actants to gain a certain degree of agency (Jones 1998) and from the moment of attaining agency actants become independent of the actors and are thus able to take some control. This very key, interesting and even ambitious approach is worth exploring because it has the potential to make a real contribution to learning theories, open source process understanding, and to actor network theory. Such nuances would be entirely missed if a social constructivist approach had been taken in this work.

Shifting out and Shifting in: Parallel Time Dimensions

Latour (1995, p275) defines *shifting out* as 'any displacement of a character either to another space or to another time or to another character' [see Fig 5 and 6]. It is a semiotic expression and these three types of shifting out are called spatial, temporal or actorial, respectively (Latour 1988c, p5). Shifting *in* reverts one back to the 'original frame' and shifting *down* denotes 'delegation to technical objects' (Latour 1992). Shifting thus creates delegates. More than one delegate is needed to verify 'reality'. One person's reading of the occurrence, or even two for that matter, would not be enough to substantiate any claim made by one. It is hoped that a third delegate would agree, at least partly, with one of the other delegates, thus confirming some event. The internal referent is what is used to compare the versions of the incident from various delegates. This step is also followed by the internal referent being compared and matched with the underwritten referent [see Fig 6], 'the internal referent of the text is complemented, asserted, evaluated by its adequation, fit, superimposition, to another referent that I will call *underwritten* (or subscribed) because it is made of another set of inscriptions that establish the credibility of the ones used in the text to establish the reference of the narration' (Latour 1988c, p13).

Shifting implies delegation and a movement between time and space. Centres of calculation are an accumulation of many traces where time and space is constructed locally; and this links with both Mol (1999) and Kavanagh and Araujo's (1995) ideas of multiplicity. Kavanagh and Araujo (1995) consider the notion of time construction using the concepts of ANT. They draw an analogy with the Japanese art of creating objects through paper folding, Origami, and how time is "networked, inscribed, folded, durable and dynamic" (Kavanagh and Araujo 1995) – what they called *chronigami*. "Time is the distant consequence of actors as they seek to create a fait accompli on their own behalf that cannot be reversed... Time does not pass. Times are what are at stake between forces. Of course one force may overtake the others, but this can only be local and temporary because permanence costs too much and requires too many allies" (Latour 1988c, p165). The Origami analogy is extended to networks and objects, "the objects themselves may be quite robust and easily retain their shape. Paradoxically, they are also very fragile and are often discarded once completed" (Kavanagh and Araujo 1995) so it is the very act of inscribing that attracts attention. Once 'durable' inscription is achieved, or irreversibility in ANT terms, then the object fades into the background, unless and until some competing actant manages to draw this attention away. It is ironic that this attention can only be regained by this object if it breaks down because smooth operation becomes something implicitly expected. It is this fragility or varying degrees of durability which allows for new associations to build and society is thus able to recursively reproduce itself (Law 1992). Latour (2002, pp248-249) explains the role played by technology in folding time and space,

> "What is folded in technical action? Time, space and the type of actants. The hammer that I find on my workbench is not contemporary to my action today: it keeps folded heterogeneous temporalities, one of which has the antiquity of the planet, because of the mineral from which it has been moulded, while another has that of the age of the oak which provided the handle, while still another has the age of the 10 years since it came out of the German factory which produced it for the market. When I

grab the handle, I insert my gesture in a 'garland of time' as Michel Serres (1995) has put it, which allows me to insert myself in a variety of temporalities or time differentials, which account for (or rather imply) the relative solidity which is often associated with technical action. What is true of time holds for space as well, for this humble hammer holds in place quite heterogeneous spaces that nothing, before the technical action, could gather together: the forests of the Ardennes, the mines of the Ruhr, the German factory, the tool van which offers discounts every Wednesday on Bourbonnais streets, and finally the workshop of a particularly clumsy Sunday bricoleur".

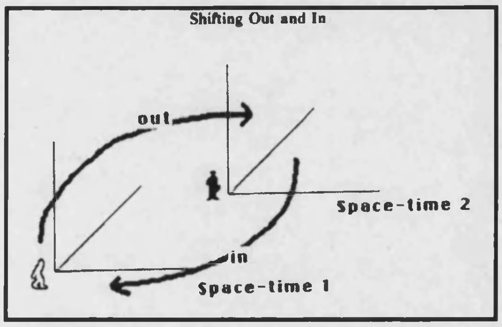


Figure 5: The two basic semiotic operations, shifting in and shifting out (Source: Latour (1988c, p6)

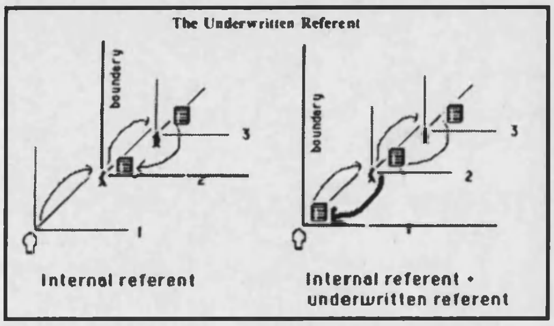

The Underwritten Referent
boundary
1
2
3
Internal referent
Internal referent + underwritten referent

Figure 6: Source - Latour (1988c, p14)

Kavanagh and Araujo's (1995) interpretation of time is interesting because according to these authors there is no universal time, "rather we see a multiplicity of times, of chronigamis, constructed in a loose, dynamic network of tangles, mangles, ensembles and assemblages". If there exist a multiplicity of times then how can there be a previously created reality which only needs to be studied to materialize? Does this not reinforce the argument for reality being constructed in the very act of actants interacting because how else or why else would there be multiple times? Latour (1987, p230) reiterates this multiplicity claim in *Science in Action* 'that space and time may be constructed locally ... these are the most common of all constructions. Space is constituted by reversible and time by irreversible displacements. Since everything depends on having elements displaced each invention of a new immutable mobile is going to trace a different space-time".

Even when software is created without a greater design developers still manage to code design *into* the very software section they are working on and thus 'speedily lock [the design] into the path leading to the final shape' (Kavanagh and Araujo 1995). This plays in neatly with Kavanagh and Araujo's (1995) chronigami metaphor. Every fold the paper is given lays down a stronger foundation for the eventual object which is created; indeed, it is the first few folds which play the most significant role in dictating what form the object will take. Can this also be said to be true of learning? Latour (1999c, p126-127) explains competence or *learning* as, "No event can be accounted for by a list of the elements that entered the situation *before* its conclusion. … If such a list were made, the actors on it would not be endowed with the competence that they will *acquire* in the event. .. They *all* leave their meeting in a different state from the one in which they entered." What we 'learn' initially dictates *how* and *what* we learn later, or which paths are 'taken' and which 'paths foreclosed' (Kavanagh and Araujo 1995). To further this, how fixed these paths become depends on the degree of inscription, just as 'macro-actors are simply micro-actors seated on top of black boxes, networks of stabilized associations between human and non-human elements' (Callon and Latour 1981, pp286-287). Such associations between humans and humans not only create a heterogeneous network but an *evolving network*, and yes it does evolve, adapt and change, in turn creating the situation in which actants themselves change. And as it is the actants that actually make up the network, when they change so does the network and vice versa.

Pickering (1993, 1995) takes up the cudgels from the epistemological chicken debate (Collins and Yearley 1992a) and through his mangle idea pushes ANT yet further. He defends ANT and its stance on material agency though other authors besides Collins and Yearley have also criticized ANT on the issue of intentionality bestowed upon material actors. Pickering, like Latour and Callon is not aiming to confer intentionality to material actors or what Schaffer (1991) calls illegitimate hylozoism. However, nor is he satisfied with the explanation given of material agency by Latour and Callon and believes that certain problems emerge when the latter authors take the agency idea from semiotics and use shifting out ideas. He doesn't agree that the material agency to human agency concept is as transferable as vice versa. His contribution comes from linking time to material agency. Pickering (1995) understands material agency as emergent temporally through practice and action. It is performative and in that moment it emerges but no actor knows what will happen next as the future is still in the process of unfolding, "human and material agency are reciprocally and emergently intertwined in this struggle.... *the dance of agency*" (Pickering 1995, p21) (italics added). In keeping with the main premise of ANT Pickering does add that human agency too is temporally emergent.

While Kavanagh and Araujo (1995) speak about the *multiplicity of time*, Annemarie Mol (1999, p74-75) attacks the same problem but her emphasis is on the multiplicity of *reality*. Mol uses the term *ontological politics*[24] to explain how, when these two words, ontology and politics, are combined, the word politics steers us away from the belief "that reality does not precede the mundane practices in which we interact with it, but is rather shaped within these practices. So the term *politics* works to underline this active mode, this process of shaping, and the fact that its

[24] A term Annemarie Mol borrowed from John Law (Law 2002)

character is both open and contested". This, according to Mol, leads us to believe that "if reality is done, if it is historically, culturally and materially located, then it is also multiple. Realities have become multiple". So, though ontological politics is 'informed by' constructivism, it 'does not directly follow from or easily coexist with... constructivism' because the emphasis is on the performative nature of interaction and thus reality is '*enacted*'. Latour thus explains the role of technology in this performance

Conclusion

Actor network theory, Bateson's levels of learning and Weick's organizing concept together make up the theoretical framework for this study. ANT offers little in the way of a definition of learning and Weick's focus is clearly on organizing (though sense-making is another way to understand learning), thus Bateson provides the necessary clarity on learning. ANT, through the concept of framing, operationalizes the link between learning and organizing. It demonstrates the performative duality between learning and organizing. This research uses the term *becoming* to describe the movement (performative duality) between varying (fleeting) states of both learning and organizing. It is becoming that draws our attention in this work and we explain this concept further through the study and analysis of the Linux kernel collective.

Chapter 5
Ordering the Performance of 'Reality'

"Tracking narratives and changes in narrative forms is a viable way of approaching the examination of meaning making and, thereby, learning in organizations... Narratives are real, everyday experience. They are the fundamental process of cognition and organizing" (Tenkasi and Boland, 1993, pp98-99).

The current study was based on an analysis of a large amount of textual data. In order to study and analyze it rigorously a content analysis of the material was carried out, and a form of discourse analysis was done. Discourse analysis is just one of many types of textual analysis that a researcher can choose from and as explained below is seen as compatible with the theoretical position of ANT. The tracing of actors and allowing them to speak for themselves is an attitude advocated by ANT and studying the discourse that leads to action and organizing is facilitated through discourse analysis, thus we can see how it is useful to approach this research with such a theoretical stance and methodological tools.

Textual Analysis

"Much of what constitutes organizational life resembles a discourse. Organizational life takes place in language. It is the process of meaning creation and of meaning sharing. Meaning generation is work in progress. Thus, as a continuous process of meaning creation and enactment the interactions which constitute organizational life may be seen as a form of organizational 'text' which may also be subject to forms of textual analysis" (Truex, 1996).

Truex (1996) explains content analysis, a form of textual analysis, as 'a search for structures and patterned regularities within the text'. Content analysis has a long history and as Krippendorff (1980) notes perhaps the first documented case of content analysis is from the eighteenth century in Sweden (Dovring, 1954-1955). The case concerned a collection of 90 hymns which were titled *Songs of Zion*. The dilemma surrounding these hymns had much to do with their content, content which was considered to be a 'carrier of dangerous ideas' and was not liked by the orthodox clergy of the Swedish state church who felt their role being undermined. This form of analysis is now used in many fields and with the growth of studies of online communities this has proved a very effective and rigorous approach of data analysis (Kollock 1999b, Wellman 2001).

Discourse Analysis

The method of research chosen was discourse analysis because this "has an analytic commitment to studying discourse as texts and talk in social practices. That is, the focus is... the medium for interaction; analysis of discourse becomes, then, analysis of what people do" (Potter 1997). It serves to clarify at this point that the method of research for this study is indeed inspired by discourse analysis but as Gill (2000) points out there are at 'least 57 varieties of discourse analysis' and it is hard to place this work as a particular kind. Rather, this researcher has adopted the techniques and tools of coding [borrowed from grounded theory] to fracture her data. Gill (2000) recommends the use of coding as one method of extracting information from empirical textual data and we found it very effective.

Discourse analysis is a 'particular type of qualitative methodology that tries to understand the processes whereby reality comes into being, rather than simply examine how actors make sense of a pre-existing reality (Hardy 2004, Phillips and Hardy 2002) . Heracleous and Barrett (2001) provide us with a framework of different schools of thought in discourse, differentiating between functional, interpretive and critical perspectives on discourse. Very briefly, the functional perspective is where discourse is understood as a 'language-based communication, used instrumentally by social actors to achieve their ends'. The interpretive perspective sees discourse as a 'communicative action, which is constructive of social and organizational reality'. Lastly, critical discourse analysis stresses 'power/knowledge relations linguistically communicated, historically located and embedded in social practice'. Phillips and Hardy (2002) add to this breakdown and Hardy (2004) furthers this work by including a distinction between a study of context and textual discourse analysis. If the focus of study is the text that is created then it is a textual discourse analysis but if the broader social context is taken into account where cultural sites and settings in which the discourse unfolds is explored then that would fall under contextual discourse analysis (Wetherell 2001). A number of other categorizations have been developed (Grant *et al.* 1998, Mumby and Clair 1997, van Dijk 1997a, van Dijk 1997b) but all are quite similar.

As Truex (1996 - 19th December) explains discourse analysis 'builds on the elementary ideas of both content and conversation analysis'. Content analysis (Krippendorff 1980, Neuendorf 2002) is basically a search for 'structures or patterned regularities within the text' (Truex 1996 - 19th December). In conversation analysis (Wynn 1979) the context of 'words' is more important because the belief here is that meaning is formed and embedded in layers of contexts and that these meanings require a process of hermeneutic analysis to be elicited. Forums are the place where OS developers interact, discuss, probe and basically live one part of their life. Discourse analysis (Klein and Truex 1996, Wood and Kroger 2000) was used because the need here is to study large amounts of text in the form of the mailing list archives of the Linux kernel developers which is their chief form of communication. The development forum of the Linux kernel developers covers both the crucial and trivial concerns. Like the narrative approach (Czarniawska 1998, Dunford and Jones 2000), discourse analysis affords the researcher with multiple versions of the same issue.

Mumby and Clair (1997) claim not to adopt discourse analysis but instead to take the concept forward and describe what they term organizational discourse. By organizational discourse they speak *not* of discourses that 'occur *in* organizations. Rather, we suggest that organizations exist only in so far as their members create them through discourse' (p181). However, they quickly clarify that they also do not mean that organizations are no more than simply discourse, rather discourse is the 'principal means by which organization members create a coherent social reality that frames their sense of who they are' a thought echoed by Doolin (2003) and Hull (1997).

Alvesson and Karreman (2000) similarly suggest that discourse 'acts as a powerful ordering force in organizations'. And organizational discourse relates the idea of how texts, or structured texts order organizational elements and bring them into being in the process by which they are created, distributed and consumed (Grant et al, 1998; Grant and Hardy, 2004; Phillips and Hardy, 2002). Discourse is understood here 'as action in society' (van Dijk, 1997) related to the idea of social action, order and organization. This link between discourse, organizing and organization is intrinsic to this study of learning in open source communities. Following Weick and Westley (1996) we see organizing and learning as a duality, where they both occur as a balance between disorder and order is achieved, but if one outstrips the other this balance is lost. And Chia (2000) explains "'organizational discourse' ... in its wider ontological sense as the bringing into existence of an 'organized' or stabilized state. Discourse works to create some sense of stability, order and predictability and to thereby produce a sustainable, functioning and liveable world from what would otherwise be an amorphous, fluxing and undifferentiated reality indifferent to causes" (p514).

The aim of this current work is to then link organizational discourse to learning and organizing, creating a framework which allows us to better understand how these concepts, when related to each other, and combined, can reveal learning and organizing in an open source collective and more generally in online communities at large. This is possible because, "Tracking narratives and changes in narrative forms is a viable way of approaching the examination of meaning making and, thereby, learning in organizations... Narratives are real, everyday experience. They are the fundamental process of cognition and organizing" (Tenkasi and Boland 1993, p98-99).

One such attempt to link organization, discourse and organizing, on which we build is Law's concept of ordering narratives. For Law (Law 1991, Law 1994) 'ordering narratives' relate how organizations change and organizing occurs. An ordering narrative takes into account the social, material and discursive dimensions of organization (Doolin, 2003). Narrative in this sense means more than just the text, written or spoken, of an organization but rather it refers to narratives that are 'recursively told, embodied, and performed in a series of different materials' (Law, 1994, p259). Doolin (2003) explains that such narratives produce 'materially heterogeneous organizational arrangements' such as machines, talk, technologies, text, people and architectures. Such narratives can thus explain how organizations are performed or enacted, understanding that organizations are not static entities, rather they are constantly being performed and there is no stability as such but just recurrent attempts at achieving and challenging stability. This is an echo of Weick and Robert's (1993) words, "narrative skills (Bruner 1990, Orr 1990a) (Weick and Browning, 1986) are important for collective mind because stories organize know-how, tacit knowledge, nuance, sequence, multiple causation, means-end relations, and consequences into a memorable plot". There is a strong link between language and learning reiterated in literature and emphasized by Weick and Westley (1996), "all learning occurs through social interaction. Language is both the tool and the repository of learning.... Language has the interesting property that it is as closely linked to forgetting as it is to learning" (p446).

Law explains that ordering narratives have certain characteristics. They are strategic, discursive, performed, materially heterogeneous and incomplete. Narratives are *strategic* because each narrative is a specific combination of the material, social and discursive, and though intentionality is not always present the fact that one combination is generated rather than any other implies a level of strategy. Ordering narratives are *discursive* because we understand the world through narratives and discourse and it is these discourses that strengthen our commitment to a worldview of reality.

Ordering narratives are *performed* and this key characteristic clarifies the link between organizing, relational performance and coming into existence. This characteristic is predicated on the idea that entities come into existence when they form some bond or relationship with other entities. Performance implies enactment, seen as a reaching out and relating to others to create a network and make you significant. As Law (1999) explains, 'entities are performed in, through and by the very relations that define them'. Such relations are with both human and non-human entities. This brings us to the next characteristic of ordering narratives, that of their *material heterogeneity*. All relations are part social and part material and it is this combination that provides durability to the relationship. Durability yes, but not long term stability or completeness, because ordering narratives are innately *incomplete*. Incompleteness refers to the fragility of any level of stability that is achieved because it doesn't last and this is why organizations are never really organizations but just constant organizing.

In his recent work Law (2004) built on his idea of ordering narratives, with 'modes of ordering'. This entire work, called *After Method: Mess in Social Science Research*, is concerned with making explicit some of the flaws in methods of research in social science. Modes of ordering is what we understand can be called an inscription device that he uses to explain the key argument in the book, that the method of research that is adopted to conduct research, in the process of its use, actually creates reality. And indeed, there is not one reality but a multiplicity of them, "Method is not, I have argued, a more or less successful set of procedures for reporting on a given reality. Rather it is performative. It helps to produce realities" (Law, 2004, p143). Law calls this the *method assemblage* and builds on this idea in his book and makes his conclusions as above but we have started with the end so a tracing of this inscription is needed here for better explanation. Drawing heavily on Latour, Callon and Mol, Law (2004) shows how researchers, using methods, probe the environment and actors for answers. The aim usually is to tell a narrative that is singular in flow and logic, and that contains *the* truth. But he stresses, how can we be so sure that there is only one true representation of reality when there has to be more than one reality because every actor, human and non-human, has their own version of reality. This is the idea behind multiplicity (Mol 2002) which exposes that there is more than one reality and that all realities usually overlap. The overlap is important because this is often what is caught by the researcher, this is the narrative that emerges from research. A particular reality emerges from the silencing or Othering of certain phenomenon and here Law plays with ideas of presence and absence. An actant's reality is unique in that it will have a particular presence of phenomenon which necessarily entails the absence of others. A researcher, when confronted with a situation, usually takes some time to listen to the emerging overlap of truth because at first it is not easy to distinguish the 'noise' from *the* overlapping reality that usually emerges. "*Realities grow out of distinctions between 'right' and 'wrong' patterns of similarity and difference*" (Law, 2004, p110 – authors italics) because according to Law, researchers look for patterns of similarity and difference and this helps them explain how they understand a situation to be, but they need to be able to find the right pattern so that "what had been dazzle, an overwhelming out-thereness, was converted into signal on the one hand and silence (which did not resonate with the relevant pattern) on the other... Bits and pieces in those observations became instances of repeatable patterns and signs of ... discursive reality of the laboratory and its ordering. At the same time other bits and pieces became less significant. The signal grew against a growing background of silence. Indeed, in due course I found it difficult to attend to forms of talk which did not fit this basic pattern of repetition" (Law, 2004, p111).

The idea of ordering narratives and modes of ordering are very similar except for the fact that ordering narratives are how what is being studied emerges, and modes of ordering is concerned with how a researcher draws out a singular story from all the noise that ordering narratives create. Both are performative and indeed they share all the characteristics listed for ordering narratives above. So one is about *what* is being studied and the other about *how* we study.

Research Design

The primary data for this research was gathered from the Linux kernel mailing list archive [LKML]. This site is kept up to date by the University of Indiana and claims to include every email message passed by the Linux kernel developers. The message on the site says, "The linux-kernel list is a majordomo mailing list hosted at vger.kernel.org. It exists for the discussion of kernel development issues, including new features, bug reports, and announcements of new kernel releases. This list is not for the faint of heart. It is designed for people who have some experience with the kernel and are interested in participating in the development of the kernel"[25].

This site was chosen, though a number of other host sites exist for kernel development, because it was a more complete site dating as far back as June, 1995. Its Google powered search allows easy access to particular threads or phrases, though many sites have also taken up Google searching, the University of Indiana site was one of the first. It has a simple and very clear interface where every year can be broken down into months and then weeks. At times other LKML sites were used to refine searches and points of reference because each site offered some unique facilities which proved helpful when cross-checking that all the material had been collected and nothing on the topic of version control been left out from the data set.

The Mailing List ARChives[26] offers similar facilities to the one held by the University of Indiana but the interface is dark and rather stark in appearance. A useful characteristic of this site was the breakdown of the number of messages sent back and forth for a single thread. This provided an easy and accurate way to measure the level of interest in a particular topic at a glance. The search facility, however of this site only scanned thread titles and not the text of emails so the results of any search were quite skewed. Another website which has a simple but attractive interface is the LKML.org[27]. This site allows one to see how many total messages were passed on all topics in a year and then breaks them down monthly, and even daily. There is a search facility on the main page but this search was not as productive of accurate results as the search engines on other host

[25] http://www.uwsg.indiana.edu/hypermail/linux/kernel/info.html
[26] http://marc.theaimsgroup.com/?l=linux-kernel
[27] http://lkml.org/lkml/

sites. A number of irrelevant emails would be grouped with useful ones because the search did not narrow down on keywords. The main page does hold some interesting information that is presented in an easy and quick to grasp fashion, for example, there is a chart which indicates the level of messages passed over a week or more and what days the developers were most active emailers.

The Linux kernel messages website held on Gossamer Threads[28] has an advanced search feature which proved very useful when trying to search for very specific terms and keywords. However, the messages held on this website only date as far back as January 1st, 1998 and the usability of the site is limited because navigating from one year to another or even month to month is not easy since it holds all the messages on pages and there is no simple way to navigate to a particular time period of messages. On the other hand it does open up entire threads for you so that you can read the conversation in a chronological flow rather than having to open up each email separately, and even better is the fact that it shows and allows access to all attachments. Many developers send their patches as attachments and thus the possibility to study these attachments is useful.

The last of the sites that merits an introduction is the Linux-Kernel[29] site. The best feature of this

[redacted]

site is its graphical display of how threads and message
s are connected and how one thread lead to
the offshoot of others. This feature was very helpfu
l in this research because the search for
messages related to version control had to be mined fr
om thousands of messages spanning many

[redacted]

are not heavily debated but the time and effort spent on negotiating the 'correct' version control software for Linux began as early as 1995, and continues to this day [the only variation being that different tools are discussed in different time periods]. Over the eight year period of Linux development under study here more than 440000 messages were sent through the LKML, and the version control discussion only makes up a little over 3000 of them (less than 1%). However, few other topics lasted or spread over so many years or even managed half the interest level that version control discourse did, [interest is measured here by the number of posts sent on a particular topic].

Taking all the threads and messages discussing version control tools from June 1995 to June 2003 a systematic collection was made, systematic but within the bounds of theoretical sampling. Theoretical sampling is defined by Glaser and Strauss (1967, p45) as "the process of data collection for generating theory whereby the analyst jointly collects, codes and analyzes his data and decides what data to collect next and where to find them". The collection was stopped in June 2003 artificially, as version control continues to be discussed today. However, we needed a cut off time in order to move away from data collection and onto data analysis. Eight years worth of data already yielded more information than strictly needed for one study, and prior to June 1995 the Linux kernel project was hardly large enough to even merit discussion of this topic.

The University of Indiana website divides the email threads and messages by week. It is kept up to date by the Unix Systems Support Group at Indiana University. However, from June 1995 to February 1996, because the message postings were fewer, the time period of each link begins by being monthly and then falls into a fortnight of messages pattern. By March 1996, the number of messages had begun to grow so from then till the present day each link on the site represents a weeks worth of threads and messages.

Data was collected in chronological order, not because this was a serious concern when it came to the analysis stage, but for the pragmatic reason of reducing the likelihood of missing any relevant threads. Often one thread will break into another one which has a different title, and unless you follow the 'story' you can easily miss an important exchange. Each link on the LKML page hosted by the University of Indiana was searched by keywords. The exploratory search introduced us to certain names of developers and keywords to begin the real search.

The LKML site allows the messages on each page to be sorted three different ways, date, subject and author wise. This facility was helpful, especially the *subject* sort because this broke all the messages down into their respective threads, thus making it possible to download each message related to every identified thread. All the messages were copied and pasted into a text document in readiness for analysis with the help of content analysis software, Atlas.ti. The researcher continued this process for all the messages in the time period of June 1995 – June 2003. The URL's were saved for each email to ensure that the researcher could return to the original text should the need arise.

The search keywords were constantly adapted as the story unfolded. Some of the keywords like CVS dropped out of use for a few years, to be taken over, in more ways than one, by BK. The criteria for a message to be included into the data set was to have some mention of version control, be it of some specific software like BK or SubVersion, or about not being able to manage without any such aid, like the thread of messages titled 'Linus doesn't scale'. This necessarily entailed reading a number of threads which potentially could have been of use, but were later discarded for their lack of direct contribution to this study. Over time the data could be said to have been saturated 'whereby no additional data' could be found where the researcher could develop more properties' (Glaser and Strauss, 1967, p61). Still, it was an organizational narrative and if a gap was felt and the developers seemed to discuss something that had not been followed by the researcher then a retracing of messages was carried out to ensure that nothing had been missed. The different steps taken to ensure that all the relevant threads and messages were collected include simple repeated searches with keywords, the use of derivative keywords, cross-searching across other LKML archive sites because their search methods offered specific facilities, and following up a number of emails from the key protagonists even with seemingly unrelated thread titles just to ensure they don't refer back to other themes of discussion.

Table 2 illustrates the total number of threads selected as related to version control over the 8 year time period of this study, with useful statistics. All 3,352 email messages were collected in a text file and then using Atlas.ti to aid with the content analysis, the next stage of data analysis began, that of coding it.

No of threads on VCS	**249**
Total messages sent	**3352**
Maximum no of messages for longest thread	**320**
Average no of messages per thread	**13**

Table 2 Number of threads and messages collected from the LKML

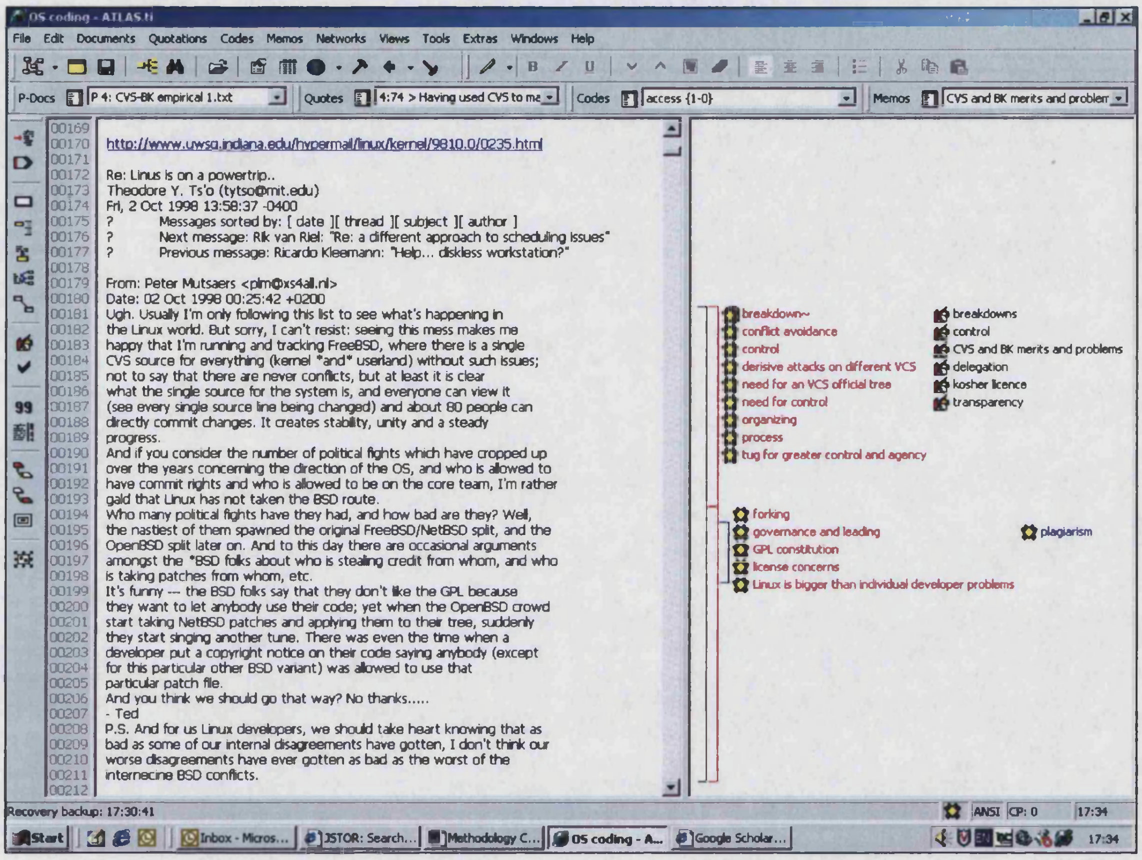
Data Analysis

All 3352 messages were saved in a text file which was then used as a primary document for the Atlas.ti content analysis software. Note however that the aim of this work is not to go down the grounded theory route but to simply use the tools of grounded theory (Strauss 1987, Strauss and Corbin 1999), in particular coding at various levels in order to fracture and expose the main themes in the data. However in this case, as explained in Chapter 7 and 8, the themes used for this work are founded on ANT and learning/organizing concepts.

A further pilot study was undertaken at this point. The purpose of this pilot was to generate a code scheme which could then be refined and made more sophisticated before returning to the body of total messages. For this a hundred messages from the over 3000 were chosen at various time periods during the eight year study period. These one hundred messages were then coded, using two levels of coding suggested by Strauss (1987), namely open coding, and axial coding. The data was never taken to the final stage of coding, selective coding, because this process involves 'integrating and refining the theory' (Strauss and Corbin, 1999, p143), and theory building in this manner is not the reason why coding was adopted in this study. Strauss and Corbin (1999) clearly lay out that if 'theory building is the goal of the research project, then findings *should* be presented as a set of interrelated concepts, not just a listing of themes' (p145) however, the method embraced here was to help themes emerge from the data, and to create theory in this way was never the intention, thus the empirical data collected was coded using only open and axial techniques. However the potential do some future work in this area has certainly proposed itself after the coding was completed, and in the former it is clear that line-by-line coding is generative of new thoughts and concepts.

Pilot Study

Coding is to allow the data 'to speak' (Strauss and Corbin, 1999, p65) and this was the purpose behind choosing this technique. Actor network theory claims that in order to understand the network the actors must be heard and traced. Coding thus becomes here the tool by which the Linux developers [amongst other actors] can be heard and followed. The pilot study hundred messages were open coded, and this gave rise to 97 initial different codes. Some were generated by the researcher, keeping in mind the theoretical framework and others [most] emerged from the data as coding proceeded. Open coding is the 'analytic process through which concepts are identified and their properties and dimensions are discovered in data' (Strauss and Corbin, 1999, p101). It is a process of fracturing the data and there are three main ways to open code data, line-by-line analysis, analyzing entire paragraphs or sentences and finally by studying entire documents. We began with line-by-line analysis of the 100 messages because we were learning to code at this stage and felt it best to start detailed and then progress to larger sections of code as we gained experience. Strauss and Corbin (1999) point out that it is this type of open coding which generates the most in-depth results and recommend this approach for any initial study noting that once a code scheme has been built up it can then be used on similar material as a guiding force.


OS coding - ATLAS.ti
File Edit Documents Quotations Codes Memos Networks Views Tools Extras Windows Help
P-Docs P 4: CVS-BK empirical 1.txt
Quotes 4:74 > Having used CVS to ma
Codes access {1-0}
Memos CVS and BK merits and problem
Recovery backup: 17:30:41
ANSI CP: 0 17:34
Start Inbox - Micros... JSTOR: Search... Methodology C... OS coding - A... Google Scholar... 17:34

```
http://www.uwsg.indiana.edu/hypermail/linux/kernel/9810.0/0235.html

Re: Linus is on a powertrip..
Theodore Y. Ts'o (tytso@mit.edu)
Fri, 2 Oct 1998 13:58:37 -0400
?         Messages sorted by: [ date ][ thread ][ subject ][ author ]
?         Next message: Rik van Riel: "Re: a different approach to scheduling issues"
?         Previous message: Ricardo Kleemann: "Help... diskless workstation?"

From: Peter Mutsaers <plm@xs4all.nl>
Date: 02 Oct 1998 00:25:42 +0200
Ugh. Usually I'm only following this list to see what's happening in
the Linux world. But sorry, I can't resist: seeing this mess makes me
happy that I'm running and tracking FreeBSD, where there is a single
CVS source for everything (kernel *and* userland) without such issues;
not to say that there are never conflicts, but at least it is clear
what the single source for the system is, and everyone can view it
(see every single source line being changed) and about 80 people can
directly commit changes. It creates stability, unity and a steady
progress.
And if you consider the number of political fights which have cropped up
over the years concerning the direction of the OS, and who is allowed to
have commit rights and who is allowed to be on the core team, I'm rather
gald that Linux has not taken the BSD route.
Who many political fights have they had, and how bad are they? Well,
the nastiest of them spawned the original FreeBSD/NetBSD split, and the
OpenBSD split later on. And to this day there are occasional arguments
amongst the *BSD folks about who is stealing credit from whom, and who
is taking patches from whom, etc.
It's funny --- the BSD folks say that they don't like the GPL because
they want to let anybody use their code; yet when the OpenBSD crowd
start taking NetBSD patches and applying them to their tree, suddenly
they start singing another tune. There was even the time when a
developer put a copyright notice on their code saying anybody (except
for this particular other BSD variant) was allowed to use that
particular patch file.
And you think we should go that way? No thanks.....
- Ted
P.S. And for us Linux developers, we should take heart knowing that as
bad as some of our internal disagreements have gotten, I don't think our
worse disagreements have ever gotten as bad as the worst of the
internecine BSD conflicts.
```

breakdown~
conflict avoidance
control
derisive attacks on different VCS
need for an VCS official tree
need for control
organizing
process
tug for greater control and agency

breakdowns
control
CVS and BK merits and problems
delegation
kosher licence
transparency

forking
governance and leading
GPL constitution
license concerns
Linux is bigger than individual developer problems

plagiarism

Figure 7: Screenshot of data analysis in Atlas.ti

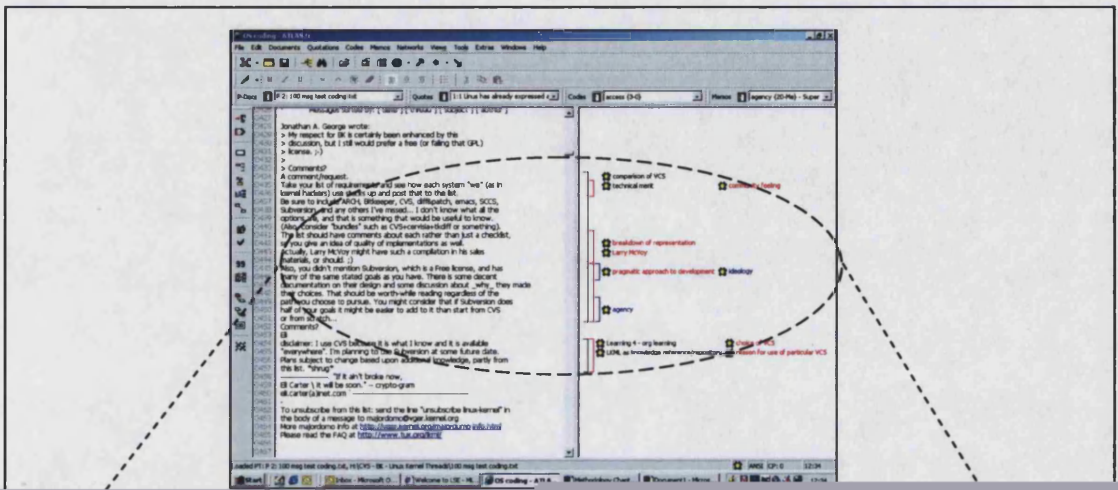
As Pries-Heje et al (2004, pp51-54) point out there are two fundamental parts to open coding: the first is of labelling phenomena; and second to discover categories. Labelling, as can be seen by the example given below is the giving a name to certain ideas that emerge as the researcher reads through the collected data. Discovering categories is the process of looking or finding relationships between different codes and data which can then be grouped together. Memo writing is one facet of this procedure.

Thus part of the process of detailed open coding of the 100 messages involved writing memos and comments about ideas that the data gave rise to or some point that the researcher needed to focus on. Memo writing in Atlas.ti was the initial attempt to move to axial code the material; axial coding implies collating some of the more detailed open codes into fewer categories. Axial coding is 'the process of relating categories to their subcategories' and is 'termed "axial" because coding occurs around the axis of a category, linking categories at the level of properties and dimensions' (Strauss and Corbin 1999, p123). The purpose of axial coding, according to Strauss and Corbin (1999, p124) is to reassemble 'data that were fractured during open coding'. There are a few steps that need to be covered for axial coding (Strauss, 1987) and they include fracturing the data, labelling their attributes, categorizing the numerous conditions, actions/interactions, and consequences related to the phenomenon and finally, looking for relationships between categories and subcategories and searching for such relationship cues in the data.

There is however, a controversy concerning different approaches to axial coding (Kendall 1999). Glaser and Strauss (1967) jointly developed grounded theory principles and methods but then went their separate ways. Glaser (1987) went on to write *Basics of Grounded Theory Analysis* (1992) and Strauss began work with Juliet Corbin (1999). The controversy was over various coding stages. Glaser (1992) developed two levels of coding, substantive [which was similar though not exactly the same as open coding] and theoretical, whereas Strauss and Corbin's (1999) coding process had three levels, open, axial and selective. Again selective coding, to a degree, resembles Glaser's theoretical coding so the real difference was axial coding. The rationale behind the introduction of this stage of coding for Strauss and Corbin was to help researchers doing grounded theory for the first time. Glaser (1992), criticized Strauss's writings, claiming that 'he fractures the concept (theoretical sampling) and dilutes its meaning by defining open sampling, relational and variational sampling and discriminate sampling, all of which occur anyway, I believe, and offer no methodological help' (p102). Robrecht (1995) made similar comments on Strauss and Corbin's work, 'the newly enlarged methodological procedures have tended to encourage the production of grounded theory with poorly integrated theoretical explanations resulting from violations of the original premises of the grounded theory method, in which theory comes directly from data' (p171). Coyne (1997) believes this encourages researchers to look *for* data rather than look *at* data (p627) and Kendall (1999) too decided to adopt Glaser's approach because she found the Strauss and Corbin way limited. Having understood how to do grounded theory using Strauss and Corbin's approach Kendall was able to progress to a more difficult but more generative of theory approach, that of Glaser's. However, the purpose behind the use of Strauss and Corbin's approach for this research is based on pragmatic choice, this is the first time, and also because Grounded Theory is not the appropriate way to link with the way this work has developed.

A Step by Step Illustration of Coding

This section takes us through the steps of how the data was analyzed using Atlas.ti software. A small section of the data is extracted here in order to be able to illustrate the example [see Fig 8 below]. The analysis was carried out using version WIN 5.0 of Atlas.ti.


OS coding - ATLAS.ti
File Edit Documents Quotations Codes Memos Networks Views Tools Extras Windows Help
P-Docs P 2: 100 msg test coding.txt
Quotes 1:1 Linus has already expressed
Codes access (3-0)
Memos agency (20-Me) - Super
comparison of VCS
technical merit
community feeling
breakdown of representation
Larry McVoy
pragmatic approach to development
ideology
agency
Learning 4 - org learning
UML as knowledge reference/repository
choice of VCS
reason for use of particular VCS
Start
Inbox - Microsoft O...
Welcome to LSE - Mi...
OS coding - ATLA...
Methodology Chart
Document1 - Micros...
ANSI CP: 0 12:34

```
Jonathan A. George wrote:
> My respect for BK is certainly been enhanced by this
> discussion, but I still would prefer a free (or failing that GPL)
> license. ;-)
>
> Comments?
A comment/request.
Take your list of requirements and see how each system "we" (as in
kernel hackers) use stacks up and post that to the list.
Be sure to include ARCH, Bitkeeper, CVS, diff&patch, emacs, SCCS,
Subversion, and any others I've missed... I don't know what all the
options are, and that is something that would be useful to know.
(Also consider "bundles" such as CVS+cervisia+tkdiff or something).
The list should have comments about each rather than just a checklist,
so you give an idea of quality of implementations as well.
Actually, Larry McVoy might have such a compilation in his sales
materials, or should. ;)
Also, you didn't mention Subversion, which is a Free license, and has
many of the same stated goals as you have. There is some decent
documentation on their design and some discussion about _why_ they made
their choices. That should be worth-while reading regardless of the
path you choose to pursue. You might consider that if Subversion does
half of your goals it might be easier to add to it than start from CVS
or from scratch...
Comments?
Eli
disclaimer: I use CVS because it is what I know and it is available
"everywhere". I'm planning to use Subversion at some future date.
Plans subject to change based upon additional knowledge, partly from
this list. *shrug*
--------------- "If it ain't broke now,
Eli Carter \ it will be soon." -- crypto-gram
eli.carter(a)inet.com `------------------------------
-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at http://vger.kernel.org/majordomo-info.html
Please read the FAQ at http://www.tux.org/lkml/
```

Loaded PT: P 2: 100 msg test coding.txt, H:\CVS - BK - Linux Kernel Threads\100 msg test coding.txt is familiar to them indicating that any new tool used implies a learning curve which means time 'wasted'. He also says that CVS is available "everywhere" and I feel that everywhere here gives the impression of likening it to air or anything else that is that freely available. Open source products are public goods but this seems to be a particular charm of CVS, along with the fact that it is a dependable tool. However this user does say that SubVersion, once it has developed further as a VCS tool will be his future choice and SubVersion is an open source tool'.

The axial codes that emerged later on did so through a consideration of the open codes and then an attempt to compress anything that appeared as a duplicate, for e.g. breakdown of representation, breakdown of translation, and breakdown, became the broader but important category of simply breakdown. Atlas.ti software allows you to link open codes in the Code Manager and even define what kind of a link you want to set up (see Fig 9). The approach of axial coding advocated by Strauss and Corbin (1999) stresses that the process of creating axial codes is carried out through 'linking categories' and by naming the relationship i.e. is one code a part of another, a cause, or contradicts the other, and so forth.

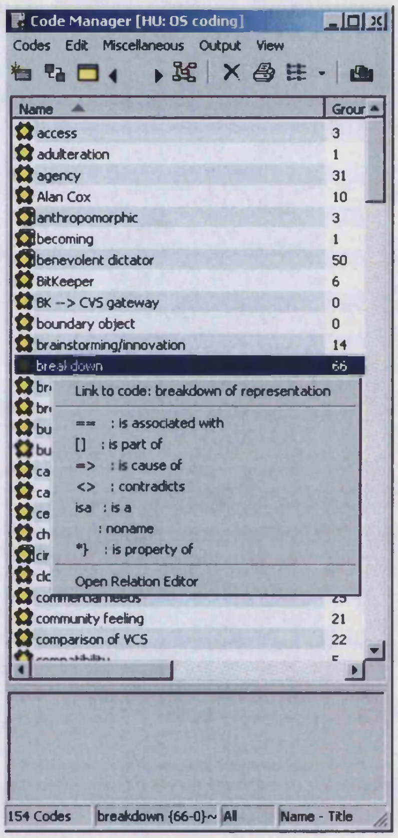


Figure 9: Linking of Open Codes to give rise to Axial Codes

Code Manager [HU: OS coding]

Name	Grour
access	3
adulteration	1
agency	31
Alan Cox	10
anthropomorphic	3
becoming	1
benevolent dictator	50
BitKeeper	6
BK --> CVS gateway	0
boundary object	0
brainstorming/innovation	14
breakdown	66
bri	
bri	
bu	
bu	
ca	
ca	
ce	
ch	
cir	
clc	
commercial needs	25
community feeling	21
comparison of VCS	22

Link to code: breakdown of representation

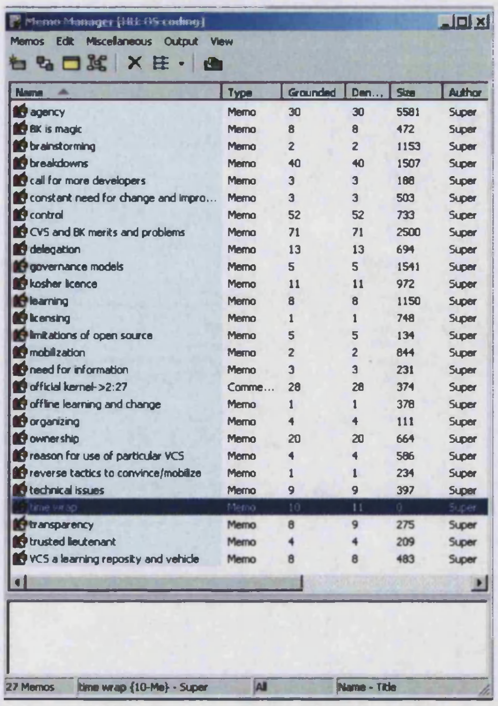
== : is associated with
[] : is part of
=> : is cause of
<> : contradicts
isa : is a
: noname
*} : is property of

Open Relation Editor

154 Codes | breakdown {66-0}~ | All | Name - Title

Data Analysis: Coding of Complete Data

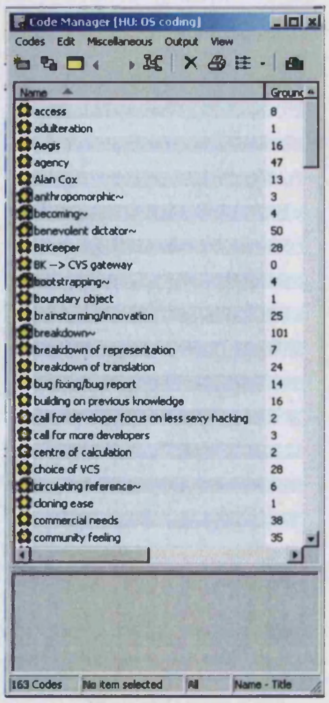
The initial coding of the 100 messages gave rise to a number of open and axial codes, and armed with these the researcher then began to code the larger bulk of her data. This time the open coding was made slightly less time consuming because now that the detailed line-by-line analysis had been done in the pilot study she was able to open code larger sections [see Fig 12]. Strauss and Corbin (1999) agree that coding entire sentences or paragraphs is a useful approach 'when the researcher already has several categories and wants to code specifically in relation to them' (p120). This is not to say that more and new open codes were not generated during the open coding of the full email data collected but the coding scheme created during the pilot study was used to guide the analysis in a large way. Indeed the open codes grew from 97 to 163 [Fig 11] (the aim was to keep the codes to a limit of 150 if possible, as having too many was beginning to defeat the purpose of organizing the data and it was generating far too many unrelated concepts. They will be very useful for a follow-up study but too distracting for this research).


Memo Manager [HU: OS coding]
Memos Edit Miscellaneous Output View

Name	Type	Grounded	Den...	Size	Author
agency	Memo	30	30	5581	Super
BK is magic	Memo	8	8	472	Super
brainstorming	Memo	2	2	1153	Super
breakdowns	Memo	40	40	1507	Super
call for more developers	Memo	3	3	188	Super
constant need for change and impro...	Memo	3	3	503	Super
control	Memo	52	52	733	Super
CVS and BK merits and problems	Memo	71	71	2500	Super
delegation	Memo	13	13	694	Super
governance models	Memo	5	5	1541	Super
kosher licence	Memo	11	11	972	Super
learning	Memo	8	8	1150	Super
licensing	Memo	1	1	748	Super
limitations of open source	Memo	5	5	134	Super
mobilization	Memo	2	2	844	Super
need for information	Memo	3	3	231	Super
official kernel->2:27	Comme...	28	28	374	Super
offline learning and change	Memo	1	1	378	Super
organizing	Memo	4	4	111	Super
ownership	Memo	20	20	664	Super
reason for use of particular VCS	Memo	4	4	586	Super
reverse tactics to convince/mobilize	Memo	1	1	234	Super
technical issues	Memo	9	9	397	Super
time wrap	Memo	10	11	0	Super
transparency	Memo	8	9	275	Super
trusted lieutenant	Memo	4	4	209	Super
VCS a learning reposity and vehicle	Memo	8	8	483	Super

27 Memos | time wrap {10-Me} - Super | All | Name - Title

Figure 10: The Collection of Memos Created and


Code Manager [HU: OS coding]
Codes Edit Miscellaneous Output View

Name	Grounc
access	8
adulteration	1
Aegis	16
agency	47
Alan Cox	13
anthropomorphic~	3
becoming~	4
benevolent dictator~	50
BitKeeper	28
BK --> CVS gateway	0
bootstrapping~	4
boundary object	1
brainstorming/innovation	25
breakdown~	101
breakdown of representation	7
breakdown of translation	24
bug fixing/bug report	14
building on previous knowledge	16
call for developer focus on less sexy hacking	2
call for more developers	3
centre of calculation	2
choice of VCS	28
circulating reference~	6
cloning ease	1
commercial needs	38
community feeling	35

163 Codes | No item selected | All | Name - Title

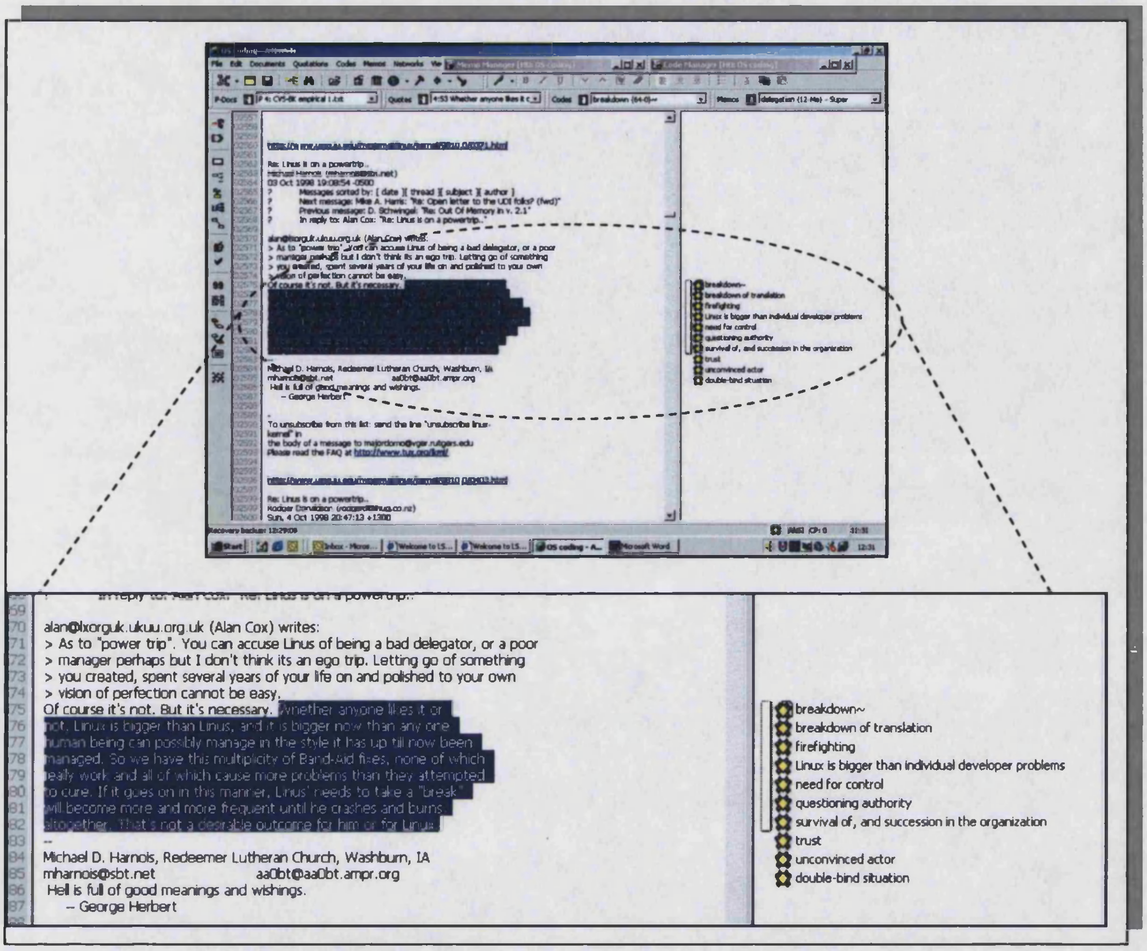
Figure 11: The Code Scheme Manager

[redacted]

tical
and
s to

that sprung to mind as the coding continued. Each memo was heavily dictated by the theore
framework of this research which is a combination of ANT, learning concepts of Bateson,
Weick's organizing ideas. Once the data was coded and memoed, the next step taken wa

[redacted]



alan@lxorguk.ukuu.org.uk (Alan Cox) writes:
> As to "power trip". You can accuse Linus of being a bad delegator, or a poor
> manager perhaps but I don't think its an ego trip. Letting go of something
> you created, spent several years of your life on and polished to your own
> vision of perfection cannot be easy.
Of course it's not. But it's necessary. Whether anyone likes it or
not, Linux is bigger than Linus, and it is bigger now than any one
human being can possibly manage in the style it has up till now been
managed. So we have this multiplicity of Band-Aid fixes, none of which
really work and all of which cause more problems than they attempted
to cure. If it goes on in this manner, Linus' needs to take a "break"
will become more and more frequent until he crashes and burns
altogether. That's not a desirable outcome for him or for Linux.
--
Michael D. Harnois, Redeemer Lutheran Church, Washburn, IA
mharnois@sbt.net   aa0bt@aa0bt.ampr.org
Hell is full of good meanings and wishings.
    -- George Herbert

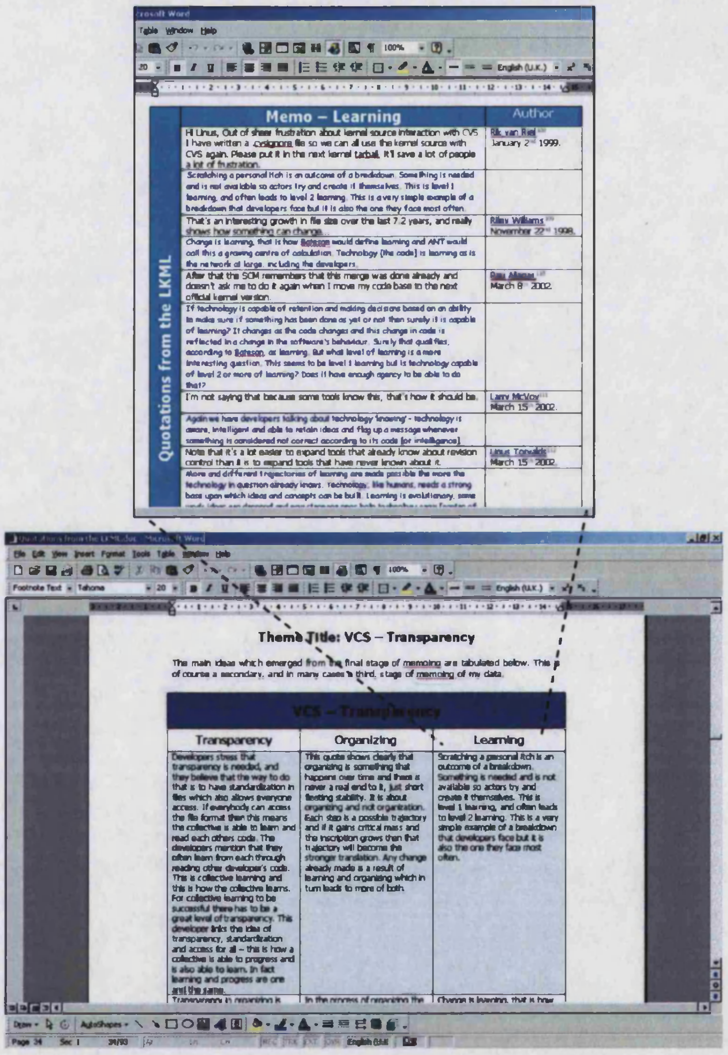
Figure 12: An Illustration of Paragraph Coding for the Complete Data

There were twenty-seven memos but the researcher felt that a few key themes were emerging from an organized grouping of some of the memos. Finally, after having studied the memos and all the notes taken during the memoing over time the researcher noticed an emergence of four main themes where the other memos and ideas began to fade into the background as noise at first, and then a simple silencing or othering (Law, 2004). The main themes indicated a strong pattern and eleven various memos helped each theme and pattern to emerge [see Table 3].

Analysis Theme	VCS - Time and Space	VCS - Material Agency	VCS – Assemblage	VCS - Transparency
Memo title	Time wrap	Agency	Mobilization	Transparency
	Learning repository	Control	GPL-Constitution	Organizing
		Ownership	Breakdown	Learning

Table 3: Main Analysis Themes and Memo Titles

The study of twenty-seven memos helped four main themes of analysis to emerge but it was only eleven memos were then extracted from Atlas.ti and copied into a word document for further scrutiny. There were over a 150 messages or message fragments which the researcher continued to work on [more than 22,000 words]. They were grouped below each memo and then a more minute and fine grained analysis of only these 150 messages was carried out [see Figure 13]. Unlike the previous coding where only some messages would generate a memo or be linked to a concept, this time each message was re-memoed more finely to generate a yet more detailed understanding of the data. The researcher reread each message but this time the aim was to see if a coherent argument and interesting revelations could be found through a linking of various memos for each theme.



Quotations from the LKML

Memo – Learning	Author
Hi Linus, Out of sheer frustration about kernel source interaction with CVS I have written a .cvsignore file so we can all use the kernel source with CVS again. Please put it in the next kernel tarball. It'll save a lot of people a lot of frustration.	Rik van Riel January 2nd 1999.
Scratching a personal itch is an outcome of a breakdown. Something is needed and is not available so actors try and create it themselves. This is level 1 learning, and often leads to level 2 learning. This is a very simple example of a breakdown that developers face but it is also the one they face most often.	
That's an interesting growth in file size over the last 7.2 years, and really shows how something can change...	Riley Williams November 22nd 1998.
Change is learning, that is how Bateson would define learning and ANT would call this a growing centre of calculation. Technology [the code] is learning as is the network at large, including the developers.	
After that the SCM remembers that this merge was done already and doesn't ask me to do it again when I move my code base to the next official kernel version.	Rau Alanas March 8th 2002.
If technology is capable of retention and making decisions based on an ability to make sure if something has been done as yet or not then surely it is capable of learning? It changes as the code changes and this change in code is reflected in a change in the software's behaviour. Surely that qualifies, according to Bateson, as learning. But what level of learning is a more interesting question. This seems to be level 1 learning but is technology capable of level 2 or more of learning? Does it have enough agency to be able to do that?	
I'm not saying that because some tools know this, that's how it should be.	Larry McVoy March 15th 2002.
Again we have developers talking about technology 'knowing' - technology is aware, intelligent and able to retain ideas and flag up a message whenever something is considered not correct according to its code [or intelligence]	
Note that it's a lot easier to expand tools that already know about revision control than it is to expand tools that have never known about it.	Linus Torvalds March 15th 2002.
More and different trajectories of learning are made possible the more the technology in question already knows. Technology, like humans, needs a strong base upon which ideas and concepts can be built. Learning is evolutionary, some	

Theme Title: VCS – Transparency

The main ideas which emerged from the final stage of memoing are tabulated below. This is of course a secondary, and in many cases a third, stage of memoing of my data.

VCS – Transparency

Transparency	Organizing	Learning
Developers stress that transparency is needed, and they believe that the way to do that is to have standardization in files which also allows everyone access. If everybody can access the file format then this means the collective is able to learn and read each others code. The developers mention that they often learn from each through reading other developer's code. This is collective learning and this is how the collective learns. For collective learning to be successful there has to be a great level of transparency. This developer links the idea of transparency, standardization and access for all – this is how a collective is able to progress and is also able to learn. In fact learning and progress are one and the same.	This quote shows clearly that organizing is something that happens over time and there is never a real end to it, just short fleeting stability. It is about organizing and not organization. Each step is a possible trajectory and if it gains critical mass and the inscription grows then that trajectory will become the stronger translation. Any change already made is a result of learning and organizing which in turn leads to more of both.	Scratching a personal itch is an outcome of a breakdown. Something is needed and is not available so actors try and create it themselves. This is level 1 learning, and often leads to level 2 learning. This is a very simple example of a breakdown that developers face but it is also the one they face most often.
Transparency in organizing is	In the process of organizing the	Change is learning, that is how

Figure 13: Screenshot indicating the more intense level of re-memoing.

As can be seen in Figure 13, each message was memoed a second time [text in blue] and these 150 or so memos were then further extracted from this document into one where now only the memos could be seen and not the data. More organizing of the concepts thus elicited from the data was then carried out. This entailed mining the memos for ideas, linking the memos together for each theme to assess for a coherent argument, and then organizing the ideas in a descending order of relevance. By relevance in this instance the researcher means those ideas that were oft repeated, revealing of something new and interesting to the researcher, or that naturally required to be put before others in order to make the following themes more consistent and logical. These organized and pain-staking steps lead to a reality emerging from the data which reflects the appropriateness of the methodology adopted for this research in practice and hopefully makes this study scientific. The analysis derived from this study is detailed in Chapters 7 and 8 but before we move to that phase of this work it is necessary that the case study and the main story of the data is related to the reader.

Conclusion

In this chapter the aim has been to explain the steps taken to perform this research rigorously. The online case study analysis of email threads and messages through coding helped develop the core themes for analysis which are fully expanded in Chapters 7 and 8.

The methodology is built on Law's ordering narratives and modes of ordering and it has steered closer to narrative and discourse analysis. The appeal and indeed the appropriateness of this strategy is reinforced by Tenkasi and Boland (1993) because they, like this researcher claim that "tracking narratives and changes in narrative forms is a viable way of approaching the examination of meaning making and, thereby, learning in organizations… Narratives are real, everyday experience. They are the fundamental process of cognition and organizing" (Tenkasi and Boland, 1993, pp98-99). This method has found a way to be folded with both learning and organizing in a manner that helps to operationalize the ability to recognize, study and analyze learning and organizing in the Linux kernel case study.

Chapter 6

The *Control* Struggle of Version Control in the Linux Collective

In this chapter we tell the story of version control adoption in the Linux kernel development collective. We allow the actors of the collective to speak for themselves and from the multitude of realities that are open to us we begin ordering one reality, the researcher's reality.

The Linux kernel development project, in the last 8 years, has engaged in a long standing controversy concerning the choice of using version control software. The present study traces this story from June 1995 to June 2003. This covers the transition from no version control software [VCS] to, first, partial use of Concurrent Versions System [CVS], and then to the use of BitKeeper [BK]. The principal source of data reported here was the Linux Kernel mailing list [LKML] archive, supported by other accounts of the period and other relevant online archives and discussion list.

The Linux kernel could probably be said to have started development around 1991 but back then it was a very small project initiated by a young Finnish student at the Helsinki University, called

[redacted]

el, it was merely to announce that "*I'm working on a free version of a minix-lookalike* system kerne

omputers. It has finally reached the stage where it's even usable ... and I am willing *for AT-386 c*

[redacted]

2002[32]]. Landley [January 28th, 2002[33]] explains further, "*He's an architect. He steers the project, vetoing patches he doesn't like and suggesting changes in direction to the developers. And he's the final integrator, pulling together dispirate patches into one big system*". However, Torvalds can inspire statements such as "*I nominate Linus for Beloved Benevolent Dictator*" (Esh, April 24th, 1996[34]) where beloved seems to be at odds with dictatorship, but he also invokes "*I wish Linus would be more responsive...Linus likes the way he does things and doesn't care if others don't like it. I don't expect to see much change there*" [Gooch, December 27th, 2001[35]]. Senior developers like Raymond say "You're our chosen benevolent dictator and maybe the second coming of Ken" [Raymond, August 22nd, 2000[36]], but also play a sobering influence if adulation and praise have, on occasion, gone to Torvalds' head "*Now you listen to grandpa for a few minutes. He may be an old fart, but he was programming when you were in diapers and he's learned a few tricks..[I] can see that *you*, poor damn genius that you are, are cruising for a serious bruising*"(Torvalds and Raymond, August 22nd, 2000[37]). This interesting character and leader has steered the Linux kernel project through many troubled moments and below is a tracing of action and actors of this collective, especially Torvalds.

Pre-VCS to Partial VCS

In the mid 1990s coordinating LINUX as a small project was simple. Of course CVS was in use by UNIX developers but again this project didn't merit the need for any such tool because it did not have critical mass yet. Torvalds was in the habit of making pre-patches and/or releases, and so coordination was achieved through individuals called maintainers who acted as a filtering device, "ftp.kernel.org is where Linus puts stuff, but I think David Miller has been putting out some fairly complex 2.0.x patches as well. I have no problem giving access to ftp.kernel.org for people working on major Linux subsystems -- please feel free to contact me" (Anvin, 1997[38]). If maintainers could keep up with patches, and not lose or confuse them, then VCS was not needed. Thus, Torvalds decided in 1995 to *not* adopt CVS for kernel development and this decision was backed by his trusted lieutenants Alan Cox and David Miller, "*I'm afraid that I don't like the idea of having developers do their own updates in my kernel source tree. I know that's how others do it, and maybe I'm paranoid, but there really aren't that many people that I trust enough to give write permissions to the kernel tree. Even people I have worked with for a long time I want to have the option of looking through their patches _first_, and maybe commenting on them (and I*

[32] http://www.uwsg.iu.edu/hypermail/linux/kernel/0204.2/1107.html
[33] http://www.uwsg.iu.edu/hypermail/linux/kernel/0201.3/1000.html
[34] http://www.uwsg.iu.edu/hypermail/linux/kernel/9604.3/0007.html
[35] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0112.3/0467.html
[36] http://www.uwsg.iu.edu/hypermail/linux/kernel/0008.2/0240.html
[37] http://www.uwsg.iu.edu/hypermail/linux/kernel/0008.2/0240.html
[38] http://www.uwsg.iu.edu/hypermail/linux/kernel/9705.2/0270.html

do reject patches from people)" (Torvalds and Schlenter 1995, 1996)[39]. As Moody describes, at least Miller was not to remain so happy with Torvalds disregarding the use of CVS (Moody 2001) and, by late 1997, CVS was widely if unofficially used within the Linux collective.

However early 1997 coordination started to become problematic. While controlling a small band of developers working on a small application was fairly straightforward, it did not help a project of the size that the Linux kernel had grown too. Questions concerning version control began to proliferate and there was doubt amongst the community about the status of CVS-use which some developers attempted to address, "*The CVS tree is not the official kernel tree. Linus controls what gets into the official kernel tree, a lot of different developers have write-access to the CVS tree*" [Rosenkraenzer, July 26th, 1998[40]]. A number of senior maintainers in late 1996 began to voice their need of some version tool. Managing all the patches being sent to them and maintaining communications within the broader community was getting impractical. Torvalds, when asked about a version tool, showed interest but was decidedly against the use of CVS. This decision of his provoked a small but persistent outcry in the community, where some said that "*I envy Alan, Linus, and Marcelo for having the ability to silently drop patches and wait for resends.*" [King, December 27th, 2001[41]] and even Rik van Riel criticized the maintainer abilities of Linus, "*Silently dropping bugfixes on the floor is not maintainership*" [van Riel, December 27th, 2001[42]]. Trust is a key feature of the OS process, the deterioration of which could lead to a breakdown of the community and process. There was even some pleading from developers to, "*Linus: Please please please please please sync the official tree up with vger, I and many others can't use CVS, and it seems that it is increasingly the only way to get a functional kernel with framebuffer support*" (Leas, 28th Sept 1998)[43]. The developers wanted Torvalds to pay attention to their patches and to provide them with one definitive tree to pull from. Messages in early 1997 proved to be the precursor to a heated debate over the decision of Torvalds to not use CVS, and September 1997 saw David Miller's post to announce "*This is a new thing I'm going to start doing", a decision to make 'full raw snapshots of my tree on vger available for ftp on a somewhat regular basis from now on. It contains also a copy of the full ChangeLog file from my CVS repository, it goes real far back, back to 1.99.x something"* (Miller 1997 - Sun, 14 Sep)[44]. CVS then had an ally close to the top, and became more acceptable within parts of the Bazaar. Torvalds himself was never

[39] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/9602/0800.html
[40] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9807.3/0161.html
[41] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0112.3/0447.html
[42] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0112.3/0498.html
[43] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0550.html
[44] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/9709.1/0432.html

convinced of CVS use, indeed was hostile to it[45], so the 'tree' kept up to date by Miller was only quasi-official.

Torvalds had his reasons for not adopting CVS for the kernel, "*I'm _really_ disappointed by how the thing has been handled, and very very disappointed by vger. The reason I'm disappointed is that vger in particular has been acting as a "buffer" between me and bug-fixes, so that now we're in the situation that there are obviously bugs, and there are obviously bug-fixes, but I don't see it as such, I only see this humongous patch. I don't know what it fixes, because vger has kept me out of the loop, and quite frankly I don't have the time to look at several hundred kilobytes of compressed patches by hand. And I refuse to apply patches that I don't feel comfortable with. As a result, I want people to _tell_ me what the patch fixes. And that probably means by now that I need to get each driver update on its own, with explanations. This is _exactly_ the same thing that made me hate vger when it came to networking patches. And I'm going to ask David once again to just shut vger down, because these problems keep on happening*" (Torvalds, 1998 – Tues 29th Sept)[46].

Miller's reply to Torvalds was just as aggressive, "I won't disable it... I want to have the problem fixed as much as the next person... People are maintaining this code, and if you block patches they can't do their job. I ask you to be a part of the solution instead of the problem" (Miller, 1998 – Tues 29th Sept)[47] and told Torvalds to "stop pointing fingers towards "vger", people are sending you patches, continually, and are being ignored and not being told why" (Miller, 1998 – Tues 29th Sept)[48]. This conversation does not disprove the fact that Miller had great respect for Torvalds and he knew that he was "David "on Linus's shitlist" Miller" (Miller, 1998 – Wed 30th Sept)[49]. Torvalds reply was not too gracious, "*I'll also take a few days off. Quite frankly, I just got very fed up with a lot of people. David, when I come back, I expect a public apology from you. Others, look yourself in the mirror, and ask yourself whether you feel confident that you could do a better job maintaining this. If you can, get back to me, and maybe we can work something out*" (Torvalds, 1998 – Tues 29th Sept)[50]. Other developers in the community recognized the fact that, "there are basically three people that should stick their heads together and have a little talk, namely Linus, Dave and Geert. If they don't, and if everybody else keeps on this "bazaar-

[45] see http://www.uwsg.indiana.edu/hypermail/Linux/kernel/9809.3/0766.html where Torvalds said, "I'm _really_ disappointed by how the thing has been handled, and very very disappointed by vger."
[46] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9809.3/0766.html
[47] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0672.html
[48] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0674.html
[49] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0877.html
[50] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0837.html

fighting", this will be the end of linux real soon" (Welwarsky, 1998 – Wed 30th Sept)[51]. There was real support for CVS in the Linux kernel collective,

> *"- One central CVS tree, with mirrors around the world.*
> *- Team of "core" members around the world (with Linus as a chairman) with write access to central CVS tree. This team decides what "official" Linux is. From other hand this team gets patches from "mortals" and add it to "official" Linux if it is "good"*
> *- All others have read-only access to CVS tree - so they could synchronize via Internet their local Linux source tree with "stable" or "development" Linux kernel branches. This also provides easy way to document what has been done and easy to convert diff descriptions to HTML or other formats"* (Indenbaum, 1998 – Oct 1st)[52].

It was during this time that a number of other solutions were proposed to help Torvalds and the Linux kernel community. One of them was a web based remedy in use by the WINE community, "*I think it would help matters all around if there were a web page somewhere. This is what Wine uses: there's a public web page for patches, but everything has to be accepted by the principal maintainer (Alexandre Julliard) to get into the main release. Some of the subsystems are maintained by other designated maintainers, who then submit patches to Alexandre. It seems to work quite well. The main tree is released on a regular fixed schedule, every two weeks except when Alexandre takes a holiday. I don't know if it would help Linus to work that way*" (Lokier, 1998 – Wed 30th Sept)[53]. However, more of the developers were interested in the use of a tool called Jitterbug including Miller, "*Maybe we have to come to terms with the fact that it is possible for a projects size to just require that there is some mechanism to "mechanize" the development process. This can come in two forms: 1) Something like Jitterbug, [or] 2) Letting more than 1 person be the only ones who can make direct changes to the source*" (Miller, 1998 – Wed 30th Sept)[54]. Torvalds did not agree with the use of Jitterbug either but developers asked "*Why was jitterbug closed down? If I understand correctly, it served as a patches queue, where patches were sorted as incoming/on hold/applied/..., so adding the category 'on hold until 2.3.0' for all those patches that won't go into 2.2.0 would be an option?*" (Niemann, 1998 – Wed 30th Sept)[55]. But Cox agreed with Torvalds in that, "*Yes the current system is a complete mess. Yes it needs fixing,*

[redacted]

Solutions aside, it was discussed that the reason why Torvalds faced more problems now than he had previously was because, "*Linus's strategy for making his "job" manageable is to make sure that only good patches get it. So he wants to make sure that the overall structure of Linux is aesthetic AND EXTENDABLE! However, patch maintainers tend to submit patches that are only LOCALLY aesthetic (if that), and may have bad implication for the OVERALL structure... So, the solution that LInus has proposed is basically that other people shoulder the work by writing better patches and thinking more about things in terms of overall structure and code structure, so that Linus doesn't have to do all the thinking about whether the patches are good for Linux's overall structure, or whether the changes will cause problems further down the road*" (Redelings, 1998 – Thurs 1st Oct)[57].

Though nothing official had been announced as yet about BitKeeper, McVoy often threw in some minor comments about the work he was doing. He wanted feedback from the developers as to what would be acceptable to them in a source code management system. The developers were aware that he was busy working on closed source version software and even at that time they showed their hesitancy about such licence mixing. McVoy was eager to please this community of developers because he needed them to do the testing of his product so he went as far as to promise, "*making BitKeeper GPLed software if BitMover ever got bought*" (McVoy, 1998 – Sun 4th Oct)[58]. McVoy was not without support because many of the developers had responsibilities, "*I'd like the Linux community to support Larry in working out the details for this "hybrid" model for "open software". I and many others would like to do something like what Larry is doing. In particular, it allows *small* software groups to produce very high quality software and still pay the mortgage and send the kids to school. Let's be pragmatic, except for over-caffeinated college students, the rest of us simply cannot afford to indulge in "free/open" software with out a clever way to make a few bucks to put food on the table. I'd like to propose a topic for Eric R. (our resident pundit and deep thinker): Software as a Cottage Industry*" (Leighton, 1998 – Mon 5th Oct)[59].

Not directly related to this story of VCS adoption but closely connected is a thread of conversation on the LKML which began in November of 1998. This was called 'The history of the Linux OS', and was a small project initiated by Simon Kenyon,

> "*The History of the Linux Operating System - I'm going to start to pull together anything and everything that I can find about the history of the Linux OS. Could everybody please send me any/all of the following:*

[57] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0094.html
[58] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0441.html
[59] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0545.html

- original and significant email/new messages
- timings of releases
- biographies of contributors to the kernel, utilities
- pointers to web pages containing information
- stories/photographs of amusing and/or significant occasions
I'll put this together on a web site and try to keep it up to date. This OS is important to a lot of people and we should keep a record of what/why/how happened and when. Any and all help appreciated" (Kenyon, 1998 – Sun 22nd Nov)[60].

This need to archive and preserve the Linux history arose because there was no version control software being used and a few developers were getting anxious about losing various versions of Linux that had been created over the years. The reason why this thread has been documented here is because it sheds light on how the Linux developers understood the older versions to be important for the learning they could glean from reading the old code, "*As Andries pointed out, the old kernels are actually fairly well preserved. What is much less well-preserved is the old original distributions. That's because they are bigger, and tended to change over time, sometimes without necessarily having distinct version numbers. But something which would definitely worth*

the distributions introduced, which distributions were descended from other distributions, etc" (Ts'o, 1998 – Mon 23rd Nov)[61]. The feeling was that some version control software was required to preserve the Linux history and at the same time make it accessible to everyone, "*I've been looking into setting up a CVS tree that will contain the full kernel source history. Put them in a* *substantial step forward from CVS*" (McVoy 1999 - Sun, 21 Feb)[64]. BitKeeper [BK] was the product of McVoy meeting with Torvalds and few of the other senior kernel developers. CVS was not efficient enough for Torvalds and there was a growing crisis around Torvalds 'dropping patches' and a long thread of messages focused on the topic of 'Linus doesn't scale'[65]; indeed there was talk about the kernel forking. McVoy took this opportunity to explain how BK would resolve many of the issues Torvalds was facing. "*Linus, Dave Miller, and Richard Henderson came up to my house for dinner and we drew pictures on the floor for about 3 or 4 hours, and when we were done, Linus said "yeah, that's cool, if you build it and it works like you say, I'll use it". And I foolishly said "No problem, I've done it before, 6 months or so". That would have been around the fall of '98 I think*" (Andrews and McVoy 2002)[66]. McVoy explained that, "*I had this picture in my mind of sort of this star topology, peer to peer system, with Linus at the centre. Then the next ring of people out are his lieutenants, like Dave and Alan. Then beyond that was another ring of people, and then beyond that was another ring of people. What I visualized was that everyone would kind of hang on to somebody else, but it would be a smaller and smaller set of people the closer you got to Linus*", [Barr, 2003[67]].

McVoy was willing to create software that would closely match the needs of specifically Torvalds because he felt that, "*Linus is what makes Linux great. He's the glue that holds it all together. Without him, Linux would splinter like the BSDs have*" (Andrews and McVoy 2002)[68]. McVoy gives much of the credit for the idea behind BitKeeper to Torvalds, "*Well, for what' it's worth, the BitKeeper stuff really got rolling after Linus, Dave Miller, and Richard Henderson came by for dinner and we hashed out how the solution should work. That picture got refined through talks with people like Eric Raymond (who came up with the patch file format, or at least a comment he made prompted the design), and kernel folks like Alan, Ted, and Stephen at Linux Expo. So not only is BitKeeper not the first thing to come along, it's also not like it's just some dream I have concocted in a vacuum. It's been, and will continue to be, a process that gets changed by the needs of the primary developers*" (McVoy, 1998 – Sun 4th Oct)[69]. Torvalds was interested but didn't give a definite reply. He insisted that he needed to use the software and then make his judgement. When McVoy's company BitMover in the late 1990s did finish the first version of BK it was not Torvalds who was the first user but, "*For a couple of years before Linus ever touched it, Cort [Cort Dougan of FSM Labs and the Linux PowerPC Group] and his team used it. They jumped on it and used it when it was very early (in the development process) and were just*

[64] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/9902.2/0812.html
[65] http://www.uwsg.iu.edu/hypermail/Linux/kernel/0201.3/1000.html
[66] http://kerneltrap.org/node.php?id=222
[67] http://www.linuxworld.com/site-stories/2003/0127.barr_p.html
[68] http://kerneltrap.org/node.php?id=222
[69] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0439.html

incredibly helpful helping us debug problems. If there was some issue, they always saved the repository so we could go poking through it and try to figure out what was going on. Those guys you know they have done more for us than anybody else." (Barr and McVoy 2003)[70]

The first real mention of BK was made by McVoy on the 21st of February, 1999 when he announced that "*Most of you know we've been working on the next gen revision control for the kernel for a while now. For those of you who don't know, BitKeeper is an Open Source distributed revision control system which I claim is a substantial step forward from CVS*" (McVoy, 1999[71]). BK was not, however, universally seen as a remedy, and, for example, inspired the 'Darkstar' project; not being open source (GPLed) was anathema to many Linux developers. However, McVoy didn't accept that BK was closed source software, "*It is you, not me, that is casting things as either "open source" or "closed source". BitKeeper isn't "closed source". You can get the source and you can fix it if you want to. You can redistribute patches as long as you don't apply them, redistribute the result, and call it BitKeeper (if you want to call it* initiative of Linus Torvalds, Sebastian Kuzminsky, Theodore Ts'o and Daniel Quinlan but it never really took off the ground (Quinlan, September 13th, 2000)[75]. The discourse on this topic generated many emails but the context steered off into a comparison of CVS and BK rather than showing any genuine interest in the proposal.

Torvalds, however, didn't think the licence would pose such an issue, and in February 2002 declared it the official version tool for the kernel, "*The long-range plan, and the real payoff, comes if main developers start using bk too, which should make syncing a lot easier*" [Torvalds, February 5th, 2002[76]] and thus it was official that the Linux kernel would use BitKeeper and this too with Linus's initiative and blessing. Torvalds suffered some initial hassles using BK when he had to "*spend about a week trying to change my working habits and scripting bitkeeper... I expect to be a bit slower to react to patches for a while yet, until the scripts are better. However, some of it pays off already. Basically, I'm aiming to be able to accept patches directly from email, with the comments in the email going into the revision control history*" [Torvalds, February 5th, 2002[77]].

[redacted]

who claimed that, "*bk is not free software, but it was chosen by Linus and other developers because of it qualities as distributed source code management system.*" [Zippel, April 22nd, 2002[78]].

Torvalds believed that CVS "*allow*[ed] *automatic acceptance of patches, and positively* [illegible]" *they need to be updated. It knows. That's a benefit of having changesets, I only need to compare the ChangeSet file to know that 4 files were updated 2 were moved, and 5 were created, then I move those *portions* of those files across the wire. Other than the initial repository create (aka cvs checkout), BK *never* moves an entire file across the wire. Never means never and includes the process of deciding what to do. CVS moves whole files just to discover there is nothing to do*" [McVoy, September 11th, 2000[81]]. David Gatwood came up in strong defence of CVS with, "Faster, yes, but it still does the same thing, just in a different way that uses less bandwidth" [Gatwood, September 11th, 2000[82]] but McVoy hit back with" *BitKeeper is faster in practice because it isn't trying to talk to the CVS server for all of the operations".* [McVoy, September 11th, 2000[83]] and he showed comparative figures for time taken to carry out a null update for both "*CVS: 139.5 seconds* and *BK: 1.6 seconds*" [McVoy, September 11th, 2000[84]].

Many in the community, due to its non-GPL licence, did not like BK. Pragmatically, Ts'o suggested "*If Linus were willing to dictate from high that we were going to use bitkeeper, and that all patches had to come in as bitkeeper changelogs, then that might get us critical mass*" [Ts'o, September 13th, 2000[85]] and he was right. A volley of dissenting voices came across the LKML, "*Linux ceases to be free software when you require nonfree software to contribute it*" and this from a trusted lieutenant, Alan Cox (Cox 2000 - Wed Sep 13)[86].

The situation was exacerbated when in early October 2002 some developers noticed that the BitKeeper license [BKL] had been changed to include a new clause, '..this License is not available to You if You...develop, produce, sell, and/or resell a product which contains substantially similar capabilities of the BitKeeper Software, or, in the reasonable opinion of BitMover, competes with the BitKeeper Software'. "*This would seem to be a change which is not Open Source developer friendly* said Gall, (Gall 2002- Fri Oct 04)[87] but McVoy replied, clarifying that BitMover [BM] had left out the word 'distribute' in the clause in order to protect open source developers (McVoy 2002 - Fri Oct 04)[88]. The substance of concern about BK not being GPLed is well expressed by Molnar, "*Today the 'Linux kernel' is not the source code anymore, it's the source code plus the BK metadata... by default the data and the MetaData is owned by whoever created it. You, me, other kernel developers. We GPL the code, but the metadata is not*

[81] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0009.1/0549.html
[82] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0009.1/0560.html
[83] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0009.1/0565.html
[84] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0009.1/0530.html
[85] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0009.1/1022.html
[86] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0009.1/1076.html
[87] http://www.uwsg.iu.edu/hypermail/Linux/kernel/0210.0/1496.html
[88] http://www.uwsg.iu.edu/hypermail/Linux/kernel/0210.0/1500.html

automatically GPL-ed." (Molnar 2002 - Sun Oct 06)[89] Molnar stressed that a change in the BKL was needed which '*ensures that metadata attached to GPL-ed code is also licensed under the GPL, and creates a clearly GPL-ed repository*'.

Support [or a need to be fair] for McVoy came from what seemed like an unlikely person, David Miller. After all Miller had argued quite sharply with Torvalds over the use of CVS but Miller did admit that BitMover does not charge the Linux community anything for their use of BK, "*nobody need[s] to give Larry one dime to use BK for kernel work. And to be honest, I get better support from Larry for *free* than you'll most often get when paying some company for software support. This is a fact. When was the last time you got a ring on your cell phone 4 minutes after emailing a bug report to someone? Larry isn't Microsoft*" [Miller, October 7th, 2002[90]]. But there were others in this community who would rather that "*your* [Larry McVoy's] *company dies ASAP and bitkeeper stops poisoning air here*", [Machek, October 6th, 2002[91]].

This topic generated a heated discussion which lasted for more than 200 email messages on the

capability to
BK to pop up
n (the same
2002[92]]

his community reacted with, "*If this is a concern, it actually appears that BK has the*
"enforce" a license, in that I could make BK aware of the GPL and that would cause B
a window saying "Do you agree to this license" before the first check-in by a perso
way it asked you whether you wanted to allow openlogging)" [Torvalds, October 6th, 2

better place if people had less ideology, and a whole lot more "I do this because it's FUN and because others might find it useful, not because I got religion"' (Torvalds 2002 - Sat Apr 20)[94].

Dual VCS the BK→CVS gateway

BK slowly began to gain ground and a number of other kernel developers adopted its use. This tentative peace broke down when in February of 2003 Pavel Machek announced that he had decided to start a project called BitBucket '*[a] bitkeeper clone*' (Machek 2003 - Wed Feb 26)[95]. Not surprisingly Larry McVoy retaliated quickly, '*BitKeeper is a trademark, please don't use the BitKeeper name when describing BitBucket*' (McVoy 2003 - Sat Mar 01)[96], because saying *"'BitBucket is a GPL-ed clone of BitKeeper' ... implies that BitBucket does what BitKeeper does and nothing could be farther from the truth*", (McVoy 2003 - Sat Mar 01)[97]. McVoy's reply to Machek then began a long repertoire of emails, most poking fun at BK and McVoy's adherence to proprietary software. This was carried out in a good humoured way where developers like Alan Cox joined in too, although he made it clear that he did think Larry was right, "*Something is wrong, I agree with Larry. He should instead say "A tool for accessing bitkeeper repositories"'* [Cox[98], 2003]. Some developer comments were: "*It would be better if you stored it in BitKeeper just to piss Larry off*", said Lehmann [2003[99]] and Eckenfels [2003[100]] added but "*that would be a license violation :)*". And some rather amusing clone names cropped up like KitBeeper and even ButtCreeper!

Garzik's "SourceForge Syndrome"

Issues which grew out of this discussion were that the developer base was getting diluted by initiating too many version control projects and thus not creating any truly useful one, what Garzik called the 'SourceForge Syndrome' (Garzik 2003 - Sat Mar 01)101; recalling how a real conversation is needed before initiating any new tool (here McVoy having dinner with Torvalds and others was brought up as an example ... (Machek 2003 - Mon Mar 03)102) (Bradford 2003 -

---

[94] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0204.2/1018.html
[95] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0302.3/0931.html
[96] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/0052.html
[97] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/0057.html
[98] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0060.html
[99] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0064.html
[100] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0068.html
[101] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/0147.html
[102] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/0422.html

Sun Mar 02)103; and why projects should be encouraged to keep open source the free and open infrastructure it is supposed to be (Hellwig 2003 - Sat Mar 01)104.

During the debate over BB Garzik, who is a strong proponent of BK kept insisting that the only reason why he was against a BK clone was because, "*a BK clone would just further divide resources and mindshare. I personally _want_ an open source SCM that is as good as, or better, than BitKeeper. The open source world needs that, and BitKeeper needs the competition. A BK clone may work with BitKeeper files, but I don't see it ever being as good as BK, because it will always be playing catch-up*" [Garzik[105], 2003]. "*I call this the "SourceForge Syndrome*", where developers ask,

> "*Q. I found a problem/bug/annoyance, how do I solve it?*
> *A. Clearly, a brand new sourceforge project is called for".*

Garzik added that "*while people would certainly use it, I can't help but think that a BK clone would damage other open source SCM efforts*". Instead, Garzik wants efforts to be geared towards improving an existing SCM tool which is able to read and write BK files, and of course this seems reasonable enough as can be witnessed by the concurring replies that he received (Anvin 2003 – Sun 2nd March)[106]. However, Jeff didn't convince

ırchitected in a way that can
ε now. It pretty much fixes all
ept the CVS style of version
like the kernel. The one thing
ıerge. This (and the resulting
n Andrew, and from everyone
I've worked with, you have to
is requires architecture, and
with the proprietary tool that

ar its final stages. He spoke strongly
)[108], "*You've got at least 20 actively*
Trust me. CVS simple CANNOT do
, and will stay that way indefinitely
superior" (Torvalds 2003 - Fri Mar

ı1.html
92.html
5.html
3.html
4.html
95.html

> "*none of the current open source SCM systems are c*
> *operate like BK. I've been using subversion for a whil*
> *the problems that CVS had, AS LONG AS you acc*
> *control. That style doesn't work for non-central work*
> *BK does that makes it worthwhile is the three-way m*
> *DAG) make handling code from Alan, from Linus, fro*
> *else possible. With CVS, subversion, or any other SCM*
> *hand merge anything past the first patch. Ugh. Th*
> *(AFAIK) BitBucket is the first try at it. Compatibility*
> *does it already is a good thing.*"

Linus Torvalds only made a contribution to this discussion ne of the technical virtues of BK (Torvalds 2003 - Tue Mar 04 *developed concurrent trees with branches at different points. this. ... Give it up. BitKeeper is simply superior to CVS/SVN since most people don't seem to even understand _why_ it is*

[103] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/04
[104] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/00
[105] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/024
[106] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/026
[107] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/061
[108] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0303.0/06

07)[109]. McVoy took this as encouragement enough to give a more detailed explanation of how BK is more technically efficient and useful to the kernel developers.

Larry's *Offensive* Defence of BK

McVoy repeated the argument that none of the developers seem to be aware of just how much effort had been put into the development of BK and that was much harder than it looked,

> *"To create something like BK is actually more difficult than creating a kernel... To understand why, think of BK as a distributed, replicated, version controlled user level file system with no limits on any of the file system events which may happened in parallel. Now put the changes back together, correctly, no matter how much parallelism there has been. Pavel hasn't understood anything but a tiny fraction of the problem space yet, he just doesn't realize it. Even Linus doesn't know how BitKeeper works.. That's not a slam on Linus or Pavel or anyone else.*
>
> *Rename handling in a distributed system is actually much harder than getting the merging done...in BK's case, there can be an infinite number of different files which all want to be src/foo.c....in a centralized system it is trivial but in a distributed system you have to handle the fact that the same symbol can be put on multiple revs. Time semantics are the hardest of all. It [time] goes forwards, backwards, and sideways on you ... Again, not a problem for CVS/SVN/whatever, all the deltas are made against the same clock. Not true in a distributed system.*
>
> *It [BK] started in May of 1997, that's almost 6 years ago ...there is more than 40*

[redacted]

> *The disbelievers think that I'm out here waving the "it's too hard" flag so you'll go away."*

[McVoy[110], 2003]

This rant was continued over the next few days with Larry pitching in more heatedly, "*BK is*

[redacted]

problem because "*there _cannot_ be multiple concurrent renames that have to be merged much later (well, CVS cannot handle renames at all..)... With a central repository, you avoid a lot of the problems, because the conflicts must have been resolved _before_ the commit ever happens*", [8th March, 2003[113]].

The BitBucket debate soon converted into a technical discussion of what are the basic requirements for a version tool. The developers talked knowledgeably about changesets and file names. McVoy, not being able to suppress himself, broke into this discussion after 50 odd messages had passed, "*in reading over your stuff, you aren't even where I was before BK was started. That's not supposed to be offensive, just an observation. As you try out the ideas you described you'll find that they don't work in all sorts of corner cases and the problem is that there are a zillion of them*", but then he did allow that, "*Arch is probably closer to mimicing how development really happens in the real world, in theory, at least, it can do better than BK, it lets you take out of order changesets and BK does not. But it is light years behind SVN in terms of actually working and being a real product. SVN is almost there, they are self hosting, they eat their own dog food, Arch is more a collection of ideas and some shell scripts. From SVN, you're going to learn more of the hard problems that actually occur, but Arch might be a better long term investment*" [McVoy[114], 9th March, 2003].

BK→CVS Gateway

However, perhaps McVoy felt that he had made little headway so less than a month later he surprised developers by creating a gateway between CVS and BK (McVoy, 2003 – Tue 11th March)[115].

> "*We have a first pass of a real time gateway between BK and this CVS tree done... What we (actually Wayne Scott) did was to write a graph traversal alg which finds the longest path through the revision history which includes all tags. For the 2.5 tree, that is currently 8298 distinct points. Each of those points has been captured in CVS as a commit. If we did our job correctly, each of these commits has the same timestamp across all files. So you should be able to get any changeset out of the CVS tree with the appropriate CVS command based on dates. We also created a ChangeSet file in the CVS tree. It has no contents, it serves as a place to capture the BK changeset comments.*"

This was not exactly encouraging competing VCS but did allow CVS users more access into what was happening in Linux development. McVoy (2003 – Tue 11th March)[116] summed up his company's motivation as,

113 http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.1/0162.html
114 http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.1/0385.html
115 http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/0889.html

"... you might be wondering why we bothered. First of all... I realized that because BK is not free software some people won't run BK to get data out of BK... I simply did not anticipate that people would be that extreme... Second, our goal is to provide the data in a way that you can get at it without being dependent on us or BK in any way. As soon as we have this debugged, I'd like to move the CVS repositories to kernel.org (if I can get HPA to agree) and then you'll have the revision history and can live without the fear of the "don't piss Larry off license". Quite frankly, we don't like the current situation any better than many of you...

Another goal is to have the freedom to evolve our file formats to be better, better performance and more features. SCCS is holding us back... we have gone as far as is possible to provide all of the information, checkin comments, data, timestamps, user names, everything. The graph traversal alg captures information at an extremely fine granularity, absolutely as fine is possible. We have 8298 distinct points over the 2.5.0 .. 2.5.64 set of changes, so it is 130 times finer than the official releases. If you think something is missing, tell us, we'll try and fix it.

[redacted]

This debate even included some neutral senior developers like Alan Cox who did try to diffuse the situation by explaining that perhaps it is a better idea that developers who have a problem with the new gateway should solve the problem themselves, "*CVS can't represent it all because CVS isn't up to the job. If the rest exists as comments then its your problem to write a VCS that can extract the comment data and represent it in full*" [Cox[119], 12th March 2003]. But, as explained Roman Zippel, the 'changeset ids are missing' thus some of the valuable information needed was not accessible to all [13th March 2003[120]]. This after McVoy emphasized that, "*it is easy to write the BK to CVS exporter completely from the command line... you guys could have done this yourself without help from us because all the metadata is right there. The whole point of prs is to be able to have a will-always-work way to get at the data or the metadata, it makes the file format a non-issue*" [12th March 2003[121]]. BitMover even allowed Linux developers who had previously used [or still did use] competing tools to access converted material [Ts'o[122], 13th March 2003].

Perhaps Larry was quite justified for his exasperated comment "*none of us at BitMover would shed a tear if you moved off BK. This has *not* been a pleasant experience for us*" [McVoy[123], 12th March 2003] because "*The main thing is that the CVS server and the tarball of the CVS repository are *not* under our control. That's the only way some people are going to believe that we're not out to screw them and it would oh-so-nice to have people think that, it really would*" [McVoy[124], 11th March 2003].

The Linux developers did have a point when they said that not everything would be available to them through this gateway because Larry admitted that though '*the full revision history is preserved*': [the CVS are only] '*CVS: 110,076 deltas over all files*' whereas '*BK: 121,891 deltas over all files*' [McVoy[125], 12th March 2003]. Collins [12th March 2003[126]] was justified in saying,

> "*I wasn't far off by saying 90%. And don't tell me I can get all the data, when in fact, I can't... The kernel's revision history is always available. I get the cset emails. I can extract all the info I want manually. The problem I have is that you are going to make it so that the original files that hold this data cannot be extracted in any meaningful way without your tools. So if bitkeeper suddenly could not be used by Linus or any others, for whatever reason, we are locked out of that original dataset*"

[119] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/1150.html

Larry insisted that nothing was missing from the data and that some of the developers were just being paranoid and mistrustful, "*we actually captured 100% of the checkin information, both in data files and in the pseudo ChangeSet file, not one byte of that is lost. All we did is collapse all the branches into the longest possible straight line, which is actually for many purposes nicer than the rats nets that you get with BK*" [McVoy[127], 12th March 2003].

It may seem that there are more developers unhappy with BK or Larry but he has some staunch loyalists like Jeff Garzik and Andreas Dilger, besides others. One developer [Brandon Low[128], 12th March 2003] very pragmatically said,

> *"I'd like to say that before Linus started using BK, close to 50% of the revision data that is now saved was completely lost in the process of him merging patches by hand into his repository. I mean be realistic, do you think that Linus kept perfect track of EVERY single ' ' ';' ')' that he changed when merging a patch with minor rejects with his repo? Do you think that every single time that he made a 1 line change or merged a 1 line change that was sent to this list it was documented and recorded? I doubt it. So now we are able to get a publicly available CVS repository with close to two times the data that was ever available before, and infinitely more than was ever available to anyone outside of Linus' own head.*
>
> *I personally think that Larry has done an amazing job supporting this project and it's goals, and I will give him a big "Thanks for all your support and hard work" at this time. I think that those of you complaining about this as bitmover clearing the road to steal our data should take a long hard look at what you are really saying and consider what BK has given us that we never had before because nothing before was ever usable by Linus."*

This flamefest even inspired a mental T-shirt for one developer [John Bradford[129], 12th March 2003], which read like,

- *"Hi Larry, BK is evil, please make a BK->CVS gateway.*
- *Well... OK, I guess I can I do that without hurting my business.*
- *Larry, this BK->CVS gateway is pure evil!!!"*

Larry's rather brazen show of ownership of the metadata disturbed a number of developers, "As for the data, you are right, we don't own that. As for the metadata which makes BK work, that's ours, not yours. BK made that metadata, you did not. If you don't like those terms, convince Linus and friends to get off of BK. That would be just fine with us" [McVoy[130], 16th March 2003]. But according to Roman Zippel Larry had left out some key events so he added them himself, "You forgot to mention that some people are not allowed to use bk (without paying) and some people also don't like the invasion of privacy (unless they pay). This has never been an issue, the

[127] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/1134.html
[128] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/1128.html
[129] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/1009.html
[130] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.2/0121.html

complete version history is only available to bk users. This makes it difficult for other SCM developers to assist users in easy exchange of information with bk users or converting their information from bk into other formats. *Flame wars happened and when the dust settled, the BitKeeper folks built a BitKeeper to CVS gateway which captures the bulk of the information*, because only the bk folks have the tools and the license to produce such a product." [Zippel[131], 20th April 2003]. On the 17th March 2003 Larry announced that the BK to CVS gateway has gone live and can now be accessed and used [2003[132]]. This is where we leave the story; however, a recent developments has been the breakdown of this gateway 'due to disuse and security problems' (Anvin 2004 - Thu Jul 22)[133].

We add a postscript that describes the turn of events in April of 2005. BK was no longer available to the Linux kernel collective, and so Torvalds decided to create his own VCS, which he named GIT.

[131] http://www.ussg.iu.edu/hypermail/linux/kernel/0304.2/0859.html
[132] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.2/0219.html
[133] http://www.uwsg.indiana.edu/hypermail/Linux/kernel/0407.2/0490.html

Postscript – April 6th, 2005 – *The breakdown of BK*

After using BK for 3 years and strongly giving it support it was quite a surprise to the people following the Linux kernel development to hear Torvalds say that he would no longer use BK for the Linux development, "we've been trying to work out a conflict over BK usage over the last month or two (and it feels like longer ;). That hasn't been working out, and as a result, the kernel team is looking at alternatives" (Torvalds, 2005 - Wed 6th April)[134]. This message on the LKML was covered by every open source related online magazine. Slashdot began a poll named 'If I were Linus, I'd replace BitKeeper with…' [135] where Mind melding was the clear favourite with 37% of the votes[136].

Torvalds made it clear that he was very satisfied with BK, "the three years with BK are definitely not wasted: I'm convinced it caused us to do things in better ways, and one of the things I'm looking at is to make sure that those things continue to work. So I just wanted to say that I'm personally very happy with BK, and with Larry. It didn't

[redacted]

forced him to give up BK use (Shankland, 2005)[144] and look for an alternative. Torvalds and a number of other developers are now happily working away on Git[145] but it is not usable yet so only time will tell if the move away from BK was in the best interests of Linux.

[144] http://news.com.com/Torvalds+unveils+new+Linux+control+system/2100-7344_3-5678651.html

[145] GIT has its own discussion forum which has seen steady growth and interest - see http://www.gelato.unsw.edu.au/archives/git/

10 Longest Threads on VCS between 1995-2003, Date order, with Ranking				
Rank	**Date**	**Title of Thread**	**Description**	**No of Messages**
9	September 1998	2.1.123 and fbcon.c	What began as a small complaint about a patch soon flared into a serious argument where Torvalds took active and aggressive part. This was around the time when there were rumours of him not being able to 'scale' and being aware of this, he can be seen to demand more respect and control. However, he ends up insulting his trusted lieutenant David Miller and tops that with heavy abuse for VGER, the CVS server and tree updated by Miller. Torvalds makes a clear demand to Miller to shut VGER down as he claims it 'keeps me out of the loop' .	73
8	November 1998	The history of the Linux OS	A desire to preserve the historical evolution of the Linux operating system spurred a developer to ask the community to send him details about significant emails that were sent in the past, personal details about contributors and contributions, and even anecdotal summaries and photos. The amount of interest this request generated is a good indication of a strong desire to archive, and to even access the archive almost as organizational memory. Another thread around this time was The Linux Kernel Compilation Project [proposal]. Interestingly, CVS was brought up as a way to preserve history. McVoy, who at that time was developing BK, made an attempt to advertise his software but most of the developers ignored and continued with CVS, and even RCS ideas.	87
5	December 2001	The direction Linux is taking	This is another thread concentrated on an urgent need for a VCS for the kernel. The conversation steered into what good maintainership should be, but was later hijacked by McVoy to promote BK. Oddly enough, it ended with some developer, who had obviously not been following this thread, asking why they don't use CVS!	118
1	January 2002	A modest proposal -- We need a patch penguin	A strong need for a VCS is voiced as Linus drops patches and developers become agitated that they and their code is ignored. Linus advises the community to "don't try to come up with a "patch penguin". Instead try to help existing maintainers, or maybe help grow new ones. THAT is the way to scalability". He added that "development is done by humans" which makes quite a claim for human supremacy over technology, however, is this a true picture of the reality, or maybe it is? The latter half of the messages are focused on BK where we see McVoy make a strong attempt to enrol Torvalds into using his software for kernel development.	320

2	April 2002	[PATCH] Remove Bitkeeper documentation from Linux tree	BitKeeper documentation is inserted into the Linux Documentation directory and this caused uproar amongst those opposed to BK, but also with some BK supporters. The kernel developers want it made clear that the kernel is not too intimately linked with a closed source software like BK. Proprietary software is looked upon with distaste and extreme mistrust. The worry is that BK will contaminate the open source code of the kernel.	181
4	April 2002	BK, deltas, snapshots and fate of pre...	A continuation of [PATCH] Remove BitKeeper. where the discussion continues about ideology, proprietary software and possible adulteration of Linux code. This is the time of early BK use and the developers are keen to put up a fight to rid the kernel of it.	123
3	October 2002	New BK License Problem?	A clause is added to the BK License which makes it illegal for any developer working on a BK competing software to use BK. This forces the developers to reassess questions about 'openness'. The developers wonder if McVoy and his company intend to creep in more subtle such changes, and as trust is very slender between the two, it is not surprising that McVoy's every move is closely watched, speculated upon, and usually not given the benefit of the doubt.	144
10	October 2002	Bitkeeper outrage, old and new	Richard Stallman initiated this thread with his obvious call to the Linux kernel community to beware of BK and its proprietary license. BK supporters rallied around in defence of BK and McVoy. Their retaliation was based on the claim that Stallman, by telling other developers to not use BK, was infringing on the community's freedom to make their own choice.	70
7	February 2003	BitBucket: GPL-ed KitBeeper clone	This thread emerged from Machek's announcement of the BitBucket [BB] project, a self-professed clone of BK. BK users and supporters gathered together to crush this move through technical arguments of BK superiority. We are led into a discussion of the merits of mostly BK, but also other VCS like SubVersion and Arch. BB never materialized but managed to cause a stir with a simple announcement.	106
6	March 2003	[ANNOUNCE] BK->CVS (real time mirror)	Focus of this thread was the announcement of the BK→CVS gateway by McVoy, however, this soon degenerated into another argument between CVS and BK users. The CVS users complained that metadata written by the developers was not complete in the gateway version of CVS. This thread linked into another, rather colourfully named one, Constant BitKeeper bitching.	97

Chapter 7

Version Control Software in the Open Source Process

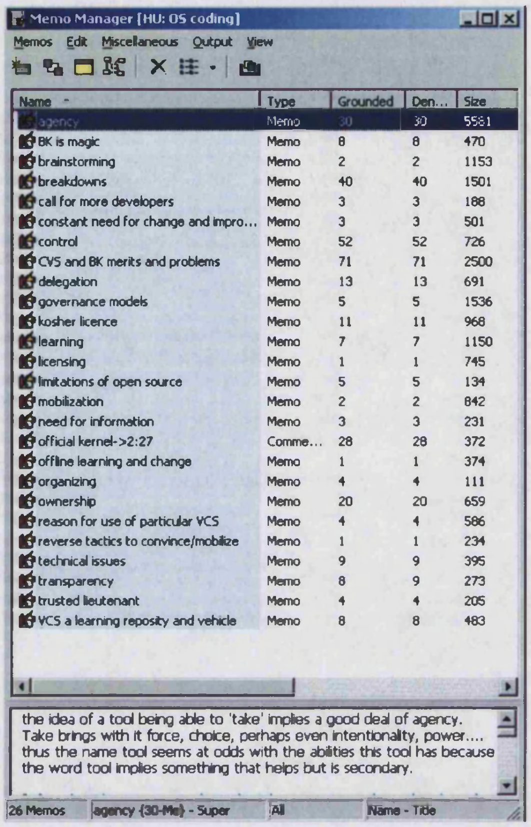
"If an organization is narrow in the images that it directs toward its own actions, then when it examines what it has said, it will see only bland displays. This means in turn that the organization won't be able to make much interesting sense of what's going on or of its place in it. That's not a trivial outcome, because the kind of sense that an organization makes of its thoughts and of itself has an effect on its ability to deal with change. An organization that continually sees itself in novel images, images that are permeated with diverse skills and sensitivities, thereby is equipped to deal with altered surroundings when they appear." (Weick)[146]

Introduction

This chapter and the following one present the analysis of our data. They are organized such that this chapter relates three themes of organizing found in the data; themes of time and space, sustaining the assemblage and material agency. The following chapter then turns to the macro theme of learning and transparency. This is not to imply that the three themes addressed in this chapter are micro in relevance, but it is structured so to provide a logical break for the reader to take pause before all themes are linked to show the overall significance of the findings.

These two analysis chapters analyze the data using the framework of ANT, Bateson and Weick developed in earlier chapters [Chapters 3 and 4]. The ontology and epistemology of ANT guided the data collection and data analysis phases of the research. The organizing of the collected data, through the use of some grounded theory techniques, gave rise to a number of open and axial

As explained in the Chapter 4, ANT, as methodology, has been operationalized here with the help of Law's concepts of ordering narratives (Law 1991, 1994) and modes of ordering (Law 2004). Both concepts stress the performative nature of reality and of doing research. Ordering narratives give sound to the voice of various actors in a collective. The emphasis is on the performative nature of narrative of the *actors*, whereas modes of ordering is concerned with how the *researcher* creates a coherent story out of the various narratives struggling for attention. The different narratives (multiplicity) have some overlap and this is often what the researcher captures, and is certainly true in this work.


Memo Manager [HU: OS coding]
Memos Edit Miscellaneous Output View

Name	Type	Grounded	Den...	Size
agency	Memo	30	30	5581
BK is magic	Memo	8	8	470
brainstorming	Memo	2	2	1153
breakdowns	Memo	40	40	1501
call for more developers	Memo	3	3	188
constant need for change and impro...	Memo	3	3	501
control	Memo	52	52	726
CVS and BK merits and problems	Memo	71	71	2500
delegation	Memo	13	13	691
governance models	Memo	5	5	1536
kosher licence	Memo	11	11	968
learning	Memo	7	7	1150
licensing	Memo	1	1	745
limitations of open source	Memo	5	5	134
mobilization	Memo	2	2	842
need for information	Memo	3	3	231
official kernel->2:27	Comme...	28	28	372
offline learning and change	Memo	1	1	374
organizing	Memo	4	4	111
ownership	Memo	20	20	659
reason for use of particular VCS	Memo	4	4	586
reverse tactics to convince/mobilize	Memo	1	1	234
technical issues	Memo	9	9	395
transparency	Memo	8	9	273
trusted lieutenant	Memo	4	4	205
VCS a learning reposity and vehicle	Memo	8	8	483

the idea of a tool being able to 'take' implies a good deal of agency. Take brings with it force, choice, perhaps even intentionality, power.... thus the name tool seems at odds with the abilities this tool has because the word tool implies something that helps but is secondary.

26 Memos | agency {30-Me} - Super | All | Name - Title

Figure 14: Final List of Memos

The four themes (narratives) of this research include; version control software and how it organizes time and space in distributed collectives; version control software and its ability to hold and sustain the assemblage; the agency of actors and how version control software is a *controlling* device; and finally (addressed in the next chapter) how version control software supports transparent learning and organizing.

Organizing Time and Space

Two main memos emerged from the data that lead to this theme of time and space. The two were titled 'time wrap' and 'learning repository'. *Time wrap*, as will become clear below, signifies that version control software is able to wrap and unwrap time and space freely and flexibly, while the *learning repository* memo helped the research to explore how the actors in the Linux collective understand the repository created by version control software as a potentially valuable learning resource which could be accessed at any point in time and space and can return you to any point in space and time.

Time Wrap: The Linux Collective's Understanding of Time and Space

Open source developers work across geographical and temporal boundaries. Neither space nor time is similar for most of the actors when for example they develop or send patches for inclusion in the Linux kernel. In open source development it is commonly understood that were it not for the creation and use of the Internet as a medium of communication then open source development would not be possible.

Here we understand the concept of time through the discourse of the Linux collective. For them time is polychronic (Lee 1999). As Lee describes polychronic time implies that 'events take place in an unexpected temporal way' meaning that events are sporadic and irregular, whereas monochronic time refers to a predetermined and predictable sequence of events. Lee distinguishes between monochronic events and a monochronic manner of working for actors. He presents a 2 by 2 matrix to indicate the various possibilities of events and actor working styles that can arise. VCS, and human actors, probably lean towards a polychronic way of working and managing events. Time is then socially and collectively constructed, and grows to be significant for actors as they communicate with each other (Lee and Liebenau 2000). In the data analyzed the collective proposes more than one perspective on both time and space, which is perhaps natural when the collective is comprised of varied actors. The most recurring awareness of time in the discourse was of two types, chronological and time measured by change. The discussion below anchors itself initially on chronological time but soon diffuses into change issues; a change in code; in behaviour; in work processes; and in perspective.

Time as efficient change: Time and efficiency are linked in the example below where the VCS that responds the fastest is considered better because time taken is a waste of resources, not just developer time, but also bandwidth use,

> *CVS: 139.5 seconds BK: 1.6 seconds In fairness to CVS, I ran this a third time and got 78 seconds, so the net must have been busy. It's still about a 50:1 performance difference. On the other hand, if you do a*
>
> *find . -type f | xargs touch*
> *time cvs update .*
>
> *it will melt down your DSL line for what seems forever. I killed it after 20 minutes, I have better things to do with my bandwidth. It's pretty clear that CVS is comparing timestamps so if your files get modified at all, it's going to transfer them to see what needs to be updated. The same sort of "touch all, then update" operation in BK*

> *has no effect on performance, BK doesn't do its work that way* (McVoy, 2000 – Sept 11th)[147].

CVS understands time through the timestamps of changes that are added to the code. It reads the timestamps on the files and if a difference is noticed it attempts to incorporate the changes. It uses a simple inscription of time. BK, as McVoy points out [see below], offers a cleverer inscription of time in its metadata and changesets. It does not work like CVS, and manages to both save time and discern a finer-grained sense of change.

> *BK only moves the data it needs to move. That means if you have a 100GB file in which you have changed one byte, BK will move on the order of 1 byte to update that file. And that's it - it doesn't compare the two files, or read the two files, or in any way look at the two files to figure out that they need to be updated. It knows. That's a benefit of having changesets, I only need to compare the ChangeSet file to know that 4 files were updated 2 were moved, and 5 were created, then I move those *portions* of those files across the wire. Other than the initial repository create (aka cvs checkout), BK *never* moves an entire file across the wire. Never means never and includes the process of deciding what to do. CVS moves whole files just to discover there is nothing to do*" (McVoy, 2000 – Sept 11th)[148].

Here time is associated directly with change and its ability to perceive and track a finer-granularity of change. And in the debate the collective relates time to technology. BK is better able to manipulate time and is thus the 'better' version control software.

In the discussion each VCS is weighed and compared based on its ability to organize, and interpret changes and to do this efficiently. The ability to interpret changes which in CVS is described as 'comparing timestamps' in BK is simply explained as 'It knows', reflecting not only the varying capacity of each as a centre of calculation, but implicitly laying claim to differing levels of learning that each VCS is capable off. The need for speed reflects the idea that VCS should use time wisely and that VCS should have the ability to interpret and distinguish changes. In other words, it needs to be 'intelligent' and be able to learn and remember as the code changes. Put another way VCS must understand change in order to control time. BK's learning is more advanced and a rather mysterious 'it knows'. It is purposefully mysterious because it is closed source and cannot afford to demystify the collective.

Time is control: Time is also understood as control. The superior abilities of BK created a situation whereby, when Torvalds adopted it for Linux kernel development, CVS not only became marginalized but was also forced to accept patches via BK, "*We have a first pass of a real time gateway between BK and this CVS tree done. Right now it is done by hand (by me) but as soon as*

[147] http://www.ussg.iu.edu/hypermail/linux/kernel/0009.1/0540.html
[148] http://www.ussg.iu.edu/hypermail/linux/kernel/0009.1/0549.html

it is debugged you will see this tree being updated about 1-3 minutes after Linus pushes to bkbits" (McVoy, 2003 – March 11th). BK is updated before CVS, albeit a few minutes apart, but BK 'knows' before CVS does and more importantly CVS cannot know unless and until BK wants it too and makes the changes available. BK thus has sought to control other VCS used by the collective. The collective needs to be kept abreast of changes, there is a need to know what every actor knows (transparency).

Time is experience: Time is also considered to be experience, "*Aegis*[149] *is under GPL and has over 8 years of development behind it*" (Mendelson, 1999 – Sept 27th)[150]. Time is measured in experience where experience is highly valued and a form of currency amongst the collective because experience is equated with learning and ability, and thus provides legitimacy. McVoy (1999 – Sept 27th)[151] makes the same point but uses BK as an example and illustrates BK embodying learning as, "*BK has...more than 8 years of proving at Sun*". Experience [gained over time] is manifested in the product, Aegis or BK. Both people and code are symmetrical in this way, for both time as experience gained is used as legitimation.

Time is process: Finally, time is seen as process, it is performative or it is becoming (a concept introduced in Chapter 3, which refers to the movement from one point to another), "*Right now, Alan's tree is in the process of going back into circulation. He tells me that his tree is basically a delta against marcello (2.4), and DJ is doing a delta against linus (2.5). Over time, the need for a 2.4 delta will probably diminish as new development shifts over to 2.5. Right now, the patch constipation we've been seeing is, in my opinion, directing development to occur against 2.4 that should at the very least be eyeing 2.5*" (Landley, 2002 – Jan 28th)[152].

The release strategy of the kernel provides some direction to the becoming, but this becoming is framed over time in a performative manner. VCS frames this performative changing for all actors to openly view. The collective is potentially given a position of advantage to view changes to software as they unfold, to see and help create the new reality together. VCS, in the process of folding current work, gives rise to a number of possible trajectories that the software could follow. In the case above growth in software development lays open more potential becomings or trajectories, but of course it siphons off other possibilities at the same time. Becoming is about possible trajectories, about potential pathways.

[149] http://aegis.sourceforge.net/
[150] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9909.3/0514.html
[151] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9909.3/0518.html
[152] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0201.3/1072.html

VCS collapses time and space: VCS provides more than just a vantage point to view reality creation (becoming), it also makes available a repository of code and code changes that allow actors to travel back in time to earlier releases and patches. So we have both a 'present' and a retrospective view of time "*if you ever wanted to know what changed in this file between 1.0.4 and 2.3.99-pre5, if you ever wanted to know at which version that file was last changed or want to find where some special structure is defined (or dozens of other things) - then you are right here :-) ok - the changes include: there is now an "all" tree which contains all versions together - so you can now diff arbitrary versions of a file*" (Graichen, 2000 – Aug 22nd)[153].

Technology as place and space: The collective is more likely to discuss time than space. Thus space, a concept closely linked to time, is less obviously evident as an area of discourse for the collective. Space is made somewhat irrelevant in open source development if the actors that participate, be they developers or circulating patches, never have to consider where they are (or others are) at any moment in time. The patches, being digital artefacts and thus weightless (Quah 2002a, 2002b, 2003), can be reproduced effortlessly and circulate, and grow as they circulate. Earlier in the kernel development the infrastructure for collaboration was discussion forums and emails but when this no longer sufficed the collective accepted another actor into its midst, VCS. We see space as folded through VCS as it brings together patches and code from developers which are based all over the world (Latour 2002). VCS is a heterogeneous mix of patches and code from all over the world, and what it manages to archive is another heterogeneous mix of actors (machination) folded together in what seems an effortless way.

VCS managing time and space: One of the reasons why a VCS is needed, besides the obvious coordination facilitation it provides, is that humans are constrained to work in chronological time and thus need help. The use of VCS provides to the human actors the ability to manoeuvre around time and space constraints, they feel they can control time through VCS use. As far as Torvalds was concerned CVS was not the correct VCS for kernel development and it threatened to take the collective down the wrong trajectory and break this assemblage of hybrid actors. A break in the collective is often not very desirable however breakdowns are an important and integral part of learning. Breakdowns bring about change if the realization that something is not working properly provokes an attempt towards improvement. Change over time happens through breakdowns, though too many can lead to increased complexity and disorganization. VCS is seen as precious and needed software because it is able to reduce the number of breakdowns and thus make experience more gainful and accessible. It provides the collective with a window on time to help deal with any breakdown and allow a quicker amelioration of the situation. Bateson (1972)

[153] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0008.2/0112.html

understands a change in behaviour to be learning but we deal with hybrid actors in this study so it is a change in hybrid behaviour. VCS manages breakdowns but also creates boundaries concerning what solutions can be adopted. It is capable of organizing time and space and in doing so it lays out possible trajectories (becomings).

Time semantics: Software is able to have an effect on the collective because it possesses agency [material agency]. For example, it makes chronological time fade away through merging and folding changes that were done in the past with recent work seamlessly, "*one important thing to consider is that time can go forward or backward. ... Time semantics. A distributed system cannot depend on reported time being correct. It can go forward or backward at any rate*" (Brown, 2003 – Mar 8th)[154]. VCS specializes in controlling parallel dimensions of time. Each branch or trajectory of software held by the VCS is a parallel dimension of time because each branch can be and is developed at the same time. VCS space is virtual where time and space are both fluid, "*You can do that, can have separate branches, distributed repository, any normal development tree can be an arch. You have to reconcile among distributed versions, star-merge, patches replay or update in any direction. You choose what you want to merge, you can always list the missing patches, you can generate the needed patches to join the branches...*" (Aliagas, 2002 – Mar 8th)[155]. Technology then makes this collective durable and does so via a passing of agency from non-humans to humans and vice versa. If you create the needed patches they will seamlessly attach themselves to the correct branch of software as if the patches themselves were looking to be part of something and contribute, or to be whole and meaningful.

Merits of Various VCS to Manage Time and Space

Ordering and efficiency: Each particular VCS manages time and space in its own way. BK is compared to CVS for kernel adoption and is declared (by some) to be technologically better and

[redacted]

How big is your revision history file? About 9,000 times bigger than a BK file doing the same thing" (McVoy, 1999 – Sept 27th)[157].

Distributed Architecture: The collective also retains an architectural dimension in its deliberations. BK is structurally more flexible and allows for greater manipulation of space and time because the architecture is distributed and thus time and space are more easily folded, "*...work includes reordering changesets/massaging them (BTW, reordering is done as adding empty changeset, pulling changes I want into it and rippling them forth; then collapsing the old one). The real difference from BK is that history and tree of changesets are independent things. It's not a "growing tree", it's "changing tree of changesets and its previous forms""* (Viro, 2001 – Dec 27th)[158]. In the discourse we see use of words like growing, rippling, and changing, all verbs that help explain that the collective sees itself as in a constant state of flux, never stable or still. These chosen words have significance, and it is interesting that the spokesman says that the code BK holds is a changing tree rather than a growing one. Change implies something far more dynamic than grow. Grow is defined as [when relating to material or immaterial things] to increase gradually in magnitude, quantity, or degree, to develop gradually (OED) but change is defined as to make (a thing) other than it was; to render different, alter, modify, transmute, to become (OED). To change does not, like to grow, mean that when something is added to X that it now becomes X + n, instead we could now have Y or even Z depending on the transmutation. This is what learning is, it changes the actor. The actor not only 'grows' but more importantly it becomes something else.

Sense-making by technology: BK does not operate like other VCS though of course its creation is inspired by other VCS (Aegis and SubVersion) not to mention to some degree CVS. But "*CVS thinks of changes as having been applied in a certain order, with each change applying to the result of previous changes. Bitkeeper does not. Each change applies to a historical version of the tree, and when it gets two sets of changes based on the same historical tree neither one of them goes "before" the other, they both apply to the old tree. (This isn't a linear process, it's lots and lots of branches. Conflicts don't come up at this point, think quantum indeterminacy and the trousers of time and all that.) So finally, when you want to know what the code looks like with all the changes added, THEN it has to figure out what order the patches have to go in to make some sort of sense. And it has to do that again the next time you ask what the tree looks like, because if you add new changes that are based on older versions of the tree, they don't go AFTER the most*

(Landley, 2003 – Jun 2nd)[159]. BK is better able to 'sense-make'. It operates like a fluid network very close to how Actor Network theory understands the world as numerous networks all connected by actor relationships (associations). Technology has control and mastery over time and space if it is *able* to make sense; be intelligent, capable of sense-making and able to learn. BK makes sense of the various changesets and is not distracted by the order in which they may have been added, it has the competence to make sense of the changes and string them together into some sensible order.

Access and visibility to aid learning: Technology can learn, but it also allows access to the collective to learn. VCS helps the collective see the entire process (framing) of software change and this is reflected in the collective changing. Through VCS the collective has access to becomings and also the fleeting stabilities achieved in the process. This process of making visible all the transactions, relationships or contact that actors undergo is called framing (Callon 1998b) but framing also makes visible all the failed relationships and transactions. The capability of VCS to support framing helps to create an open and transparent environment and is what lends to its capacity to fashion a learning-friendly environment.

The bias in the current discussion has been of time, but space is no less central to understanding VCS. Latour (2005b) explains how time and space can be folded with the example of a handkerchief. If it is spread out then the distance between two extreme corners is great but if you then crumple it up distant points suddenly become very close. VCS manipulates code and patches in a similar manner, and the superior VCS will be able to do this with more flexibility,

> "*But you can't figure out what each patch means without looking at the original, not just at the results of the previous patch. This is the fundamental problem with purely a linear approach like CVS: you either get patch 2 stomping patch 1, or vice versa, or a "it just doesn't apply" error. Creating a linear set of changes by coming up with a sane order to do it in, and adjusting later ones to take earlier ones into account, isn't that hard. But when you add more intermediate patches between point A and point B and come up with a new list, the new intermediate patches don't naturally go on the end, they go in the middle somewhere (since that's the version the changes were done against), and they may affect several of the patches after them slightly. This is why the CVS repository has to be recreated from scratch every time. It's like the difference between a typewriter and a word processor, when you insert in the middle of the paragraph everything after it gets wordwrapped differently. CVS is a typewriter that can only type at the end, and has to stick "errata" and correction notices at the end of the book you're writing to keep track of any changes you make. Bitkeeper is more like a word processor, where the order changes are made in doesn't matter so much because*

[159] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0306.0/0443.html

it doesn't have to wordwrap the sucker into its final form until you're done" (Landley, 2003 – Jun 2nd)[160].

Control of space: Space however, is not an uncontested issue. As Foucault (1995) using Bentham's Panopticon example explains, space is used to create and sustain power dynamics. Discipline "proceeds from the distribution of individuals in space... Each individual has his own place; and each place its individual.' What it avoids are 'distributions in groups' and the 'break up' of 'collective dispositions'. What it seeks are analyses of 'confused, massive or transient pluralities'. Thus, disciplinary space 'tends to be divided into as many sections as there are bodies or elements to be distributed. One must eliminate the effects of imprecise distributions, the uncontrolled disappearance of individuals, their diffuse circulation, their unusable and dangerous coagulation ... Discipline organizes an analytical space" (Foucault 1995, pp141 and 143) from Kornberger and Clegg (2004). Space is thus a highly political issue because how it is organized has important repercussions on how information and actors move through it and use it. As Kornberger and Clegg (2004) state "the exercise of power is not added from the outside, it works from the inside, it is inscribed into the heart of the spatial organization: in fact, architecture is power" (pp1104).

The visibility a VCS affords the collective of the code and the process is needed but comes at an architectural price. The CVS architecture was 'too' open for Torvalds, almost to the point of promiscuity as it meant that he had to monitor every point of entrance, of which there were many. As all actors had equal access (equal space) to add code there were additions being made from everywhere. Torvalds found it more difficult to control the process of development and identify what was happening at each stage of the Linux kernel code. BK, according to Torvalds, offered the Linux kernel collective exactly what they needed, which was open access and transparency but still gave the final decision-making authority (power) to either Torvalds or to his ally BK [through the auto merge tool]. Such access provides the opportunity and transparency needed for actants to study and reflect on what code changes happened and why, but from their own allotted space.

Learning Repository

Time and space are manipulated by VCS, and it allows tracing of process through time and space. The collective's keenness to create a historically true development can be traced back to their need for a full repository which would create a memory. This is only possible if all the code is available, open and accessible and easily retrievable.

[160] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0306.0/0443.html

Access to the History of Code

"*The most important point (for me) is that currently no history and no log messages are visible. CVS would solve this, even if only one person commits*" (Cracauer, 1998 – Oct 5th)[161]. This developer, amongst others, voiced a strong need for visibility of code and kernel history. Code is precious and has real significance for the collective because they jointly create it. When developers discuss archiving and how precious the data is they understand it as an organizational memory, for code to be open and accessible to all is encapsulated by the GPL. The freedoms or rights that the GPL endows on the licensee form the Constitution (Weber, 2004) of the collective. The GPL, coupled with Linus's Law[162] - "Given enough eyeballs, all bugs are shallow" – create an environment and ideology which propagates visibility, openness and transparency of code which can ensure future development and possibilities [possible becomings or trajectories].

Code is scarce and valuable: The urge to build a full history is strong because developers, and their time and effort is valuable. There is panic if a file is missing or gets corrupted and the collective tries to hunt for an uncorrupted version of it, "*there's actually quite a few missing from it, as well as one that's corrupted. All the ones that I've tracked down are listed in the kernel version index I've put online at the URL below, and those I've come across that aren't on the SunSite-UK mirror of ftp.kernel.org ...can be found at...* ftp://ftp.amush.ml.org/pub/rhw/Linux/ *All have been timestamped with the most recent timestamp contained therein*" (Williams, 1998 – Nov 27th)[163]. The repository is not with patches in some random order, the files are timestamped so that the order is kept intact, so it is more about which patch follows another rather, than a close binding to chronological time.

Searchable history: The developers voice a need for the code to be not only visible but browsable as well. A good VCS must provide a history, but more importantly, a searchable history. Old source code is precious because it manifests knowledge. It can be and is used as a way to learn about kernel development, revisit issues and learn, and how to code better. With millions of lines of code stored there is plenty of knowledge but there is a need to be able to browse and search it. OS developers are famous for 'scratching their own itch' so when a developer asked "*I want to know what tools can be used to browse thru the kernel source code under linux. Looking at the volume of the source code find and grep can become very annoying*" (Raju, 1999 – May 21st)[164], another quickly replied that, "*two tools that work excellent together, are bonsai and lxr (of*

[161] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0548.html
[162] http://www.free-soft.org/literature/papers/esr/cathedral-bazaar/cathedral-bazaar-4.html
[163] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9811.3/0746.html
[164] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9905.3/0067.html

www.mozilla.org and lxr.linux.no respectively). Bonsai let you query and browse CVS histories, and LXR let you cross-reference the files and every function/variable" (Wesen, 1999 – May 22nd)[165].

Visibility of breakdowns: The collective attaches real importance to code visibility because "*it would make things easier for developers to watch code changes as they happen, see where collisions with their own patches occur as the kernel progresses, and a few other things*" (Brozefsky, 1997 – Apr 10th)[166]. The increased transparency argument stems from a necessity to be able to read and study code and to critique it,

> "*the most important thing is probably to look for special cases the new code will break and that the committer haven't been aware of. A single person with a wide overview may be good in this area, but it is not scalable. Also, a patch that was committed, but backed out after protests may still be of value, since it:*
> *- Documents that this way doesn't work (and why), better than a mailing list archive can.*
> *- The patch may still be useful for some people and they can find it in the RCS files.*
> *Personally, I climb up the walls everytime I look at a Linux kernel patch because I don't have commit messages for the diffs. They're *so* handy*" (Cracauer, 1998 – Sept 30th)[167].

The above message extract captures some central ideas discussed in this research. It demonstrates that the collective is sensitive to the idea that breakdowns create awareness. This extract very nicely brings together the idea that VCS can be a repository for the Linux kernel collective, not just a historical repository but a learning repository. Past learning is manifest through the code but also in the way that VCS makes it transparent for all to see and study. Code is to be studied, read, and critiqued and the learning from this process continues as it is poured back into the collective.

Conclusion

This analysis of the 'organizing time and space' theme crystallizes three main ideas:

Managing space and time: Time is understood by the collective variously to mean change, control, experience and process. Such a polychronic understanding provides insight regarding the role VCS might perform in the collective. VCS should allow for change and make changes easily apparent. Time is also measured in experience or vice versa, and finally VCS is a window to watch processes of change. Time is inscribed into artefacts (bugfixes and patches).

[165] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9905.3/0067.html
[166] http://www.ussg.iu.edu/hypermail/linux/kernel/9704.1/0125.html
[167] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0898.html

VCS displaces control in the collective: The ability to organize and manipulate time and space makes VCS a strong actor. In the Linux kernel case it becomes apparent that CVS displaces to a degree Torvalds' control and this becomes the motivation for Torvalds to search for another VCS. Each VCS has some strengths and merits but the deciding factor in favour of BK adoption, for Torvalds, is the ability of BK [and its push and pull changes technique] to persuade the collective that they control the code.

Time/space and VCS repository creation: The idea of a repository of kernel code and time and space are inseparable from each other. Code is rare and precious, and thus there is a need to safeguard it, but what makes it more valuable is the open source principle of reusing code.

Sustaining the Assemblage

A fundamental question in this research concerns how the collective manages to hold together and sustain itself - to appear as organized. What makes the mobilization of actors possible is a combination of factors such as the leadership model, the role assigned to technology, the license scheme of the collective (Constitution), and how effectively breakdowns in work communication and relationships are handled. The data analysis gave rise to three main relevant memos which were grouped together to make sense of how the Linux kernel collective is sustained. These memos were *mobilization*, *GPL-constitution* and *breakdowns*. Combined together these help make sense of what kept and continues to keep the kernel collective working together as a cohesive group (becoming).

Mobilization

Capable steward: The Linux kernel assemblage has been through a number of crises since 1991 but has managed to maintain itself as a collective and mobilize enough actors to not only survive but also to grow. The leadership model of the Linux kernel is usually called a benevolent dictatorship, which is a paradox in itself, and is revealing about what the collective expect from their leader. He should be kind and generous but decisive and autocratic at the same time. More than that he should be an efficient manager and organizer of resources, but the only way he can do the latter is if the leader is able to visualize the greater scheme of where the software development is headed,

> *"Linux is what it is in very large part because it has a highly discriminating and intelligent filter (Linus) who also has the whole picture in his head. The size and sanity of the kernel are jealously guarded. Any disruption of this modality would put Linux in severe*

danger of becoming NT-ish - fixes/features going in without clear under-standing or motivation. Bloat and decay lie that way. With more companies becoming dependant on Linux's ongoing success, it would behoove "Someone(TM)" (Intel? Netscape? Some enlightened consortium?) to sponsor/hire 1 or 2 highly capable "mechanics" to do Linus's grunt work, and let him concentrate on what Linux most needs to retain - a careful, capable steward who is unwilling to let simple expedience ruin everything. These mechanics should be tools and collaboration gurus, not kernel gods. What are the chances of that happening? The current distributed volunteer method looks like it is becoming strained at the upper levels, and it behooves those who feed at the Linux trough to do something that demonstrates foresight and commitment" (Freeman, 1998 – Sept 30th)[168].

The developers are enrolled into the Linux kernel network – they may have some concerns, serious concerns at times, with the leader Torvalds, but are only slightly disillusioned at this time because they still feel the need to keep the leader appeased. Their belief at this point is that their leader Torvalds is the ultimate filter and controller for the kernel. However, he needs assistance of a technological actor, a human, or even better, a hybrid. Torvalds is seen as the obligatory passage point for not just the developers but also their patches and the collective demands some way to help him scale.

Still, VCS is not half as important as Torvalds because, unlike VCS, Torvalds can 'see' the direction of the project. The basic idea expressed here is that the developers give continued support to Torvalds no matter how much he breaks down. He has their loyalty and no competing translation is able to persuade them away from kernel development.

Patch inscription: The collective, as we know from the story of kernel development [Chapter 6], realize that as the collective grows so do Torvalds' responsibilities. But patches are dropped and ignored and the number of unhappy developers and patches grows from a murmur to a loud roar. Cracks begin to appear in the collective and there is a need for some aid to keep the assemblage from breaking apart. Being part of the collective is the way to ensure survival for the patches and developers alike. Patches lying around will be ignored and that is not what they want nor their writers.

"*We are most likely to find full source releases, not patches sitting around, and it's unlikely that interim patches for older releases (i.e. the equivalent of 2.0.35ac*) have survived. And, as you noted, even the patches aren't guaranteed to produce official released source trees....*" (Albery, 1998 – Nov 24th)[169].

[168] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0902.html
[169] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9811.3/0291.html

This post makes clear that patches become significant if they are attached to other patches, there must be interaction for them to exist in terms of functionality and this is evidence that reality is created when actors interact because any actor on its own is meaningless and ineffectual. It leaves no trace. When patches interact they create useful algorithms and thus the resulting software is valuable and there is a need to store it – the learning repository.

Assistance of a tool: The discourse in the kernel focuses on searches for options and they decide that as Torvalds is already helped by his trusted lieutenants, and it is one of them (David Miller and his vger) that is indirectly causing Torvalds some grief, that perhaps some technological actor is required but there are certain requirements it must fulfil before collective acceptance, "*this tool (or a connected one) should be able to force the user to enter all important data (by asking appropriate questions) and then redirect the report to the right forum/developer*" (Rosenbaum, 1999 – Sept 28th)[170]. We now see the rise of a minor obligatory passage point [OPP], technology provides the first filter and supports most of the key details needed, but then it too must yield and pass this information onto another actor, in most cases human. There are two interesting developments here, first of course we have the rise of the secondary but weaker OPP but, we can also observe humans passing some of their agency to material actors. Technology is distinctly called a tool and is only allowed to do what the humans make room for or design. However, this is an important step for technology because it now wields some measure of control over the collective. We had Torvalds [and a few of his lieutenants] mobilizing the collective alone but now we have Torvalds *plus* some tool.

Tool taking control to become OPP: The use of some technical help for Torvalds was only a suggestion at first but then slowly we see the emergence of more widespread use of VCS within the collective, "*I've started using BitKeeper to control Linux 2.4 source code. My latest tree can be found at linux24.bkbits.net*" (Tosatti, 2002 – Mar 13th)[171]. Software is able to control other software. This is one of Torvalds trusted lieutenants speaking on behalf of the stable branch of Linux [even numbered releases are the stable ones]. Such key actors accepting VCS was an immense inscription device for translating the goals and opinions of the collective. Marcelo Tosatti's enrolment into VCS use and the BK network is guided by the belief that it allows him to control his stable branch of development better and make faster releases. The BK network is growing when the actors that join its network are influential, like Tosatti, and bring in their own network.

[170] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9909.3/0648.html
[171] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0203.1/0642.html

VCS is life-giving blood for the collective: BK is able to build on its strengths and in its keenness to grow stronger is willing to offer the Linux kernel collective every update free of monetary charge. This is an effective strategy, not only because BK is free but also because it is a more able way of communicating patches and any changes to the rest of the collective. It has managed to grow in strength and that too quite quickly, "*if it was not enough that "bleeding-edge" required a pint of blood but now we get BitKeeper access to the latest and greatest. Cool! Maybe I should be on the regular shipment list from the local blood bank. Keep up the good work. I certainly appreciate it*" (Torri, 2002 – Mar 14th)[172].

The implication is that BK gives far more than it demands. Other technology, like CVS, asks for effort and manoeuvring from the developers but BK gives so much and also offers the most advanced techniques. The analogy between BK and blood is interesting. A VCS needs to be up-to-date and so software must be upgraded constantly, thus there is a need to be linked to the 'blood bank' for regular shipments. Technology is better if it is constantly learning and the centre

[illegible] growing. This is what BK is able to offer the Linux kernel collective. At this time it has managed to insinuate itself into the Linux kernel and is now busy enrolling as many actors as possible. We see here a move away from a discussion of Torvalds as coordinator and a focus on VCS, or more precisely BK. This tool has managed to create a strong enough network to sit beside Torvalds as the dual OPP, "*Through the magic of BK :) Just do a 'bk pull' on my marcelo-2.4 tree. Since it is based on the original linux-2.4 tree just like Marcelo's tree, I was able to merge from my 2.4 line to his 2.4 line*" (Garzik, 2002 – Mar 14th)[173]. As long as Torvalds is happy to have BK take charge the collective is too. The collective's new found respect is based not simply on Torvalds' acceptance of BK but also on the technical merit of BK. Merging is made effortless and thus appears magical with BK. It is not just merging of different trees or lines of development but also a merge of two different dimensions of time. Garzik is able to merge his work with Marcelo's tree which were both created in different time and space.

BK has made working and coordinating so simple that actors don't question what they lose. It is a relative feeling, because some consider the freedom to be able to do what they want with code very precious but just because they are working on OS code does not necessarily entail that they all covet freedom. For some actors being able to write code and have it included into the project is enough. In this section we have seen the collective move from a single OPP, Torvalds, to a dual one where Torvalds now shares [and happily] the OPP position with BK. It is talked of as a tool but it is a 'tool' with real power, so perhaps tool [as we conventionally consider it] is the wrong word for VCS.

---

[172] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0203.1/0903.html
[173] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0203.1/0902.html

GPL- Constitution

Law is embedded in code and is dictated by the license chosen for the software (Lessig 1999a, Weber 2004). The license lays down the Constitution for the software product and guides what the code is allowed to do and what other actors can do with, and too, the code. We do however see the collective struggle when their GPL ideology conflicts with making practical decisions. And we can see how and whom resolves this dilemma (partially at least). Finally the collective is forced to make a compromise – a compromise that seems to appease most actors if not make them entirely happy.

Ideology versus Pragmatism

FSM vs OSI: The ideological strength of Stallman's beliefs is often considered rather fanatical and obsessive and so Raymond and the OSI group attempted to put a more pragmatic twist on free software. This is not to say that they adopted more permissive licences, because they too work under the GNU Public Licence [GPL]. In this aspect they were in keeping with Stallman's original idea of free software. Stallman created the GPL because it "*protects the crucial freedoms for every user, which means that middlemen cannot pass along our code but strip off the freedom. This is the old "We're not free unless we are `free' to deny freedom to others" argument that some*

GPL vs BKL: The strength of ideological beliefs is not just restricted to Stallman. A substantial part of the collective is wary of anything that is not GPL'd, "*don't expect me to care about your pseudo-open-source product's bottom line*" (Marshall, 1999 – Sept 27th)[176]. And we can see how aggressively this section of the collective fights back when their belief system is belittled as it was by Hawkins (1998) "*You're supposed to be using the software that deals with the problem, not some half-assed GPL'd variant*" (Hawkins, 1998 – Oct 4th)[177]. Hawkins made this remark in relation to some members of the collective remonstrating against the use of BK for kernel development. The counter attack to Hawkins remark raised some real concerns with BK's licence [BKL],

> "*Supposed to be? I would think that they're "supposed to be using" software that they're comfortable with. If the licensing of a product makes me uncomfortable, I won't use it. Simple. Effective. I'm happy, and the more people who vote like me, the bigger the hint passed along to the licensee. *shrug* And I think some free software developers (myself included) would find your depiction of GPL'd software as being 'half-assed' more than a little insulting.*" (Marshall, 1998 – Oct 3rd)[178].

This interchange is reflective of how the collective is not just one generic network but many smaller networks. One such network is the staunch GPL believer one, and to this collective of actors software should be GPL'd and no consorting with proprietary code should ever be sanctioned. The GPL network is in competition with the BK network for superiority. Overall this competition may make Linux kernel software stronger because the collective is forced to understand many issues before big decisions are taken. As we have seen the collective worries about breakdowns and tries to take pre-emptive action. The worry is that BK could change Linux kernel code in a way that might make it inaccessible to the collective that created it. The license clearly dictates what can be done with the code and is linked to how open or transparent the activity is and how this transparency is reflected in the created code.

Opaque and invasive software: Not only is access to the source code restricted, but BK software is also very invasive and insidious,

> "*I got bitkeeper. It forced me to give my email address (ouch#1). Licence contains such things as "all your changes are now ours" and "if our costs on you exceed something, we may terminate your licence". You will not be even able to use your changes (ouch #2) Then I realized I'm getting self-extracting executable (ouch#3)... Self-extracting executable is about as evil as thing can be, because I am not able to say what damage it tries to do when I run it. So I created*

[176] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9909.3/0578.html
[177] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0387.html
[178] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0390.html

sandbox user (fortunately I'm root) and installed. And then... If contains

```
guest@amd:/opt/bitkeeper$ ./diff --version
diff - GNU diffutils version 2.7
guest@amd:/opt/bitkeeper$
```

That's GPL'ed tool, but you have not included text of GPL anywhere close, and I do not see written offer that you'll give me sources in the package. Even if you included those sources, I would not be able to regenerate that self-extracting archive. I think you are violating GPL by putting diff etc. into self-extracting archive user has no chance to regenerate. Issue should be easy to resolve by including GPL, offer for sources, and by distributing bitkeeper in plain .tar.gz" (Machek, 2002 – Mar 15th)[179].

Here we see that in order to make itself indispensable to the collective BK has 'written in' certain regulations and rules that must be obeyed before BK can be put to use. This is apparent with the GPL as well, "*either its allowed by the GPL or its not. There are good reasons to think that may ways of doing it are not (The GPL defines source as including installation instructions). However that's a debate for lawyers, and you can have the debate as long as you like but it doesn't change what the GPL says*" (Cox, 2003 – Apr 24th)[180]. Alan Cox makes it clear that Torvalds is not the boss of the code, it is the law [Constitution] embedded within the code that dictates what is allowed. The law is clear and must be obeyed as it stamps all code produced under its patronage.

BK provides an example of code taking control of any actor that interacts with it. The way code mobilizes actors (e.g. on installation) whether open source software is used or proprietary, is not always so obvious but at least with open source software, because the source is open and transparent the interacting actor is in a better position to make a more informed decision about adopting the software. BK, because it is closed source software, is invasive. It is working towards maintaining its position of indispensability (OPP). Its lack of transparency creates a shroud that helps it to manipulate and manoeuvre the collective. Many Linux kernel developers are experts who use the transparency and openness of the source to keep abreast of changes and to learn from their peers. GPL offers developers the ability to replicate certain algorithms, it is thus more scientific as it is open to scrutiny. BK is using some GPL'd code but that means that all of BK should become GPL'd, however, BK has used some debatable loopholes in the law to protect itself from having to give its source to anyone, or so Machek claims. Perhaps this can be explained because the GPL has not been enforced (GPL has never been defended in court (Moglen 2001)).

[179] http://atrey.karlin.mff.cuni.cz/~pavel/bitkeeper.txt
[180] http://www.ussg.iu.edu/hypermail/linux/kernel/0304.3/0236.html

Pragmatic adoption of BK: The interesting question that arises is why some actors of the collective are accepting of proprietary software use when they obviously and eagerly work on open source software out of choice? It could possibly be because in this particular case the owner and creator of BK, McVoy, makes the collective a very enticing deal, "*It's a private company, I own 100% of it. It's likely to stay that way for exactly the reasons that you want. I would be willing to make a poison pill in the company that says if my ownership ever goes lower than 51%, the sources as of that moment become GPLed (that says nothing about the sources going forward, BitMover could do whatever they want with that), but at least you'd be assured that there would be something out there*" (McVoy, 1998 – Oct 3rd)[181]. The collective is eager to sustain the survival of Linux kernel code, and recognizing this fact the owner of BK proposes to make a deal which might remove competing translations from BK and thus encourage the collective to enrol into BK use. This makes apparent the very pragmatic side of the collective. Their belief system is important but it seems that they or at least a large enough section of the collective is willing to

[redacted]

Code is Practical

License fanaticism: In the study the clash of two different constitutions is obvious. BK brings the 'outsider' constitution inside the collective and the GPL supporters are very unhappy. A clash of political beliefs ensues between actors. The clash seems to indicate that it is dictated by economic concerns but perhaps it is more about the code, desirous of pragmatic solutions, taking charge and telling the collective what it wants. The developers are the spokesmen for patches and code, but there is an insistence on getting the job done rather than just ideology. This is interesting because when we hear trusted lieutenants like David Miller speak up for GPL'd software we are convinced that the kernel would not do well to make BK use compulsory, "*I'm not sure that it's fanaticism. How is "we won't use/ distribute/ promote this product if the license doesn't suit us" fanaticism? Also, note that a number of companies have been making money off of software with free licenses. [I'm sure you've heard of Cygnus and Red Hat.] Which merely underlines the point that making money off of software doesn't necessarily require that its use is restricted -- though it does require understanding your market*" (Miller, 1998 – Oct 3rd)[183]. Miller points out that economics play a minimal role in this argument yet we eventually see BK enrol a large part of the collective. It manages this through various means of inscription, goal translation, creating conflicts which it then comes in to solve in the light of a hero, and perhaps bribes. Most importantly, however, it seems to have greater sway over the patches than the developers because we see the patches become mobilized with the simple argument that I [BK] can find you a mate quickly and efficiently and you will no longer be ignored or deleted.

Spokesman for Linux code and BK: BK supporters couch their argument in terms of efficient and effective version control software which will be a competent coordinator of code, developers and ideas. Software is the real boss in open source development, a point made by Hawkins (1998),

> "*Beggars can't be choosers. You're supposed to be using the software that deals with the problem, not some half-assed GPL'd variant. The license Larry is proposing is quite reasonable IMO. Basically what it amounts to is a royalty fee to commercial developers using Larry's software to make money for themselves. How can that NOT suit us? We're developing free software, therefore we are not required to pay for a commercial license. I can tell you misunderstand the meaning of "free". In the context of "free software" it refers to the source code being available - hence why the term has been recently renamed "open source" to avoid confusion by the people who think it means "no cost". Money has nothing to do with it whatsoever. Last time I read the GPL, it was perfectly legal to sell software licensed under it for fun and profit with the provisos of the GPL such as you provide the full source code with it, it's allowed to be*

[183] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0325.html

> *modified, etc. You're implying that Linus conforms to your perverted view of the software community and won't use a better tool simply because it isn't GPL..... and here insult him by saying that Linus is incapable of understanding let alone resolving patch conflicts*" (Hawkins, 1998 – Oct 4th)[184].

Clearly the pragmatic approach to development is taken to be the only logical one to be adopted. It is taken to be a matter of fact [not in the Latourian sense]. Ideology is sidelined by the practicalities of development. The most efficient software is always chosen and in this case it seems that BK is winning through improved service and efficient algorithms. The writer believes that anyone dictated by ideology has a 'perverted' understanding of software development. The non-believers in idealistic OS are as extreme in their views as the GPL followers. Software, and VCS, inspire very passionate discussions about how things should be done. If the GPL followers believe closed source to be a sacrilegious way for development, the unconvinced are equally scathing about GPL'd software – 'half-assed GPL'd variant'. The Linux kernel collective is also [redacted] software development and management so that no patches are lost and the kernel grows in size and functionality. We argue that this is the code speaking through this spokesman because survival of the patches is at stake. So many licenses are similar in their intentions that the collective is getting annoyed over the fastidiousness of some actors.

More is better: The need to preserve kernel code and archive it for future reference is an oft-repeated theme in the developer discourse. The comments that make the kernel code understandable, the metadata, are often seen as just as valuable as the code. A VCS that is able to generate such comments and preserve them over time has a definite advantage over competing VCS, "*CVS is the "preferred form" of the Linux kernel source code, because it is freely available. If BK has everything in it that CVS does, and also information that is not even POSSIBLE to store in CVS (i.e. ChangeSet information which links a bunch of individual file changes and comments into a single change entity) what happens then? If you had never put the kernel into BK, that information wouldn't exist at all, yet it is not possible to extract it without resorting to some source-of-all-evil tool like BK (I hope everyone reading here understands the sarcasm, but the fact that I have to annotate it makes me believe some people will not). The fact that BK is used creates information which WOULD NOT HAVE EXISTED had BK not existed. In fact, until BK was in use by Linus, not even basic CVS checkin comments existed, so the metadata was in a format called linux-kernel mbox (if that). So, the use of a tool like BK makes more data available, but people cannot be worse off than when the kernel was shipped as a tarball and periodic patches. For the sake of those people who don't or can't use BK, just pretend BK doesn't exist and they will not be any worse off than a year ago*" (Dilger, 2003 – Jan 20th)[186]. This draws back on pragmatism. The claim is that BK makes more information available than CVS was ever capable of doing or even Torvalds when he was acting as the VCS so the question is does any VCS, be it OS or otherwise lead to greater transparency of code and metadata? The argument that OS is more transparent does hold true in theory, but in practice we find that it all depends on the ability of the software to manipulate information and provide detail. If the VCS is useful it will provide an archive, with both code and metadata, will function efficiently, and coordinate activities of the various actors in an organized fashion. Following this line of argument the licence thus does not play such a key role in determining the choice of adoption, it is efficiency and meeting the particular needs of the collective.

licences is forced to work together in the same binary there is a problem. The GPL is an aggressive and viral licence which forces other licences into oblivion.

> "*Its simply a question of intent. The GPL is intended to create software guarded by a very strict set of protections designed to keep the software "free" to a set of definitions it imposes. As part of that it brooks no compromises. It draws a firm line and labels one side GPL compatible the other side not. The BSD style licenses are either*
> *Take a copy, enjoy it, hope its useful but don't bug or blame me, or*
> *Take a copy, enjoy it, hope its useful, don't bug or blame me but I want crediting in all your documentation and advertising...*
> *And you can dual license things too. ..I'd encourage people to look at both licenses when writing stuff and if they simply want people to use their stuff, commercially, binary only, whatever then to use a dual license. If they want to enforce source access etc then the GPL is probably the right choice*" (Cox, 1998 – Oct 5th)[187].

Each license varies in what it allows the code to do, and is eyed with scepticism by the other licenses. Each has its own politics and may not allow mixing. Each is protective of its code and the code in turn uses the licence when appropriate to make itself heard. In the case of the GPL we know that any code that is under a different licence, if used with GPL'd code automatically becomes GPL'd. We can understand this as the code flexing its muscles and showing agency. Another interesting concept we see emerge here is that of the dual licence, or the ability to exist under different constitutions, or 'dual nationality'. If any pragmatic approach is preferred in OS development, it can be argued that this because the code is pragmatic and has little loyalty or belief in one constitution, and/or that developers too are pragmatic in their approach to coding. Can we also add that the constitution of code is more flexible and open than appears? The collective are able to side step it when needed, or clutch it when it seems it will ensure control and survival of a particular way of existence. The move towards dual licensing in open source is definitely worth exploring because it reflects a form of hybridization or compromise.

Technology as Excluding and Including (Collecting)

The parallel release strategy used by Linux kernel developers gives both the developers and the code more time to learn if they wish to take this process slower than other actors. New versions solve problems found in older versions. Technology is learning and changing, and so are the developers who read, use and implement the code. When code is changed the behaviour of code changes (learning) and it is now able to do something different and perhaps better. The code changes as does what holds the code, VCS, and the developers change too. It is interesting to see that some changes are completely groundbreaking for the collective while others solve smaller

[187] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0586.html

breakdowns. Perhaps this is a manifestation of levels of learning or could be seen as such through Bateson's levels of learning.

BK excluding alternative VCS users: The ability to learn varies from actor to actor but there is also another possible obstacle. Not all actors have the same right of access to collectively created software. Technology can help exclude or be including of others (Winner 1986). Technology is political just like humans and in the case of BK can be used to exclude any actor that has worked on a competing VCS or has decided not to adopt BK for use. This raises other questions, since BK is closed source software, about the possible defilement of Linux kernel software, "*Using a closed-source, proprietary source control system for the kernel is even worse than using other forms of proprietary software such as source code analysis systems, because the revision control metadata (version numbers, branches, changelog comments, etc.), would be stored in a format defined by the proprietary software. This metadata is really a part of Linux, because people will want to use it when talking about the kernel. Those who can't or don't want to use BitKeeper are left out in the cold. One of the most important parts of Open Source and Free Software is that we, the community, are in control. But by using and advocating BitKeeper, we would lose part of that control*" (OS Club at the Ohio State University, 2002 – Mar 5th)[188].

VCS, as even the name implies is version *control* software, it is there to control and part of the mechanism of control is to restrain and regulate others. BK does not allow access to everybody, even those who create Linux kernel code. If they want to submit code they have to choose another channel, but what is worse is that they may have written code but cannot access it once it has been accepted and included into the kernel. This has repercussions on the collective because they can no longer study the code and versions written previously in the same format as most of the collective, however as this spokesman mentions, what troubles them more is that the informative comments, the metadata, are strictly only accessible to BK users and via BK, and BK can keep members of the collective away from code. There is also a threat that BK may in the future decide to restrict the entire collective's access. It won't be able to restrict access to Linux kernel code, but the metadata is created by BK and this throws it's ownership into a grey area. The collective wants ownership of it to be clearer and in their favour,

> "*for years people sent emails to Linus that described patches and this was not a big issue - Linus has kept 99% of the metadata in the source code. But today the 'Linux kernel' is not the source code anymore, it's the source code plus the BK metadata, which are separate bits, and this creates a new situation. the BKL.txt license currently says:*

> *By transmitting the Metadatato an Open Logging server, You hereby grant BitMover, or any other operator of an Open Logging server, permission to republish the Metadata sent by the BitKeeper Software to the Open Logging server.*
>
> *what i'm worried about is the following issue: by default the data and the MetaData is owned by whoever created it. You, me, other kernel developers. We GPL the code, but the metadata is not automatically GPL-ed, just like writing a book about the Linux kernel is not necessarily GPL-ed. ..I'm 100% sure that the Linux commit messages are already valuable today, and they will become a few orders more valuable in a few years. btw, this is also the case with the emails Linus puts into BK commit info - the email someone sends to Linus is _not_ GPLed by default." (Molnar, 2002 – Oct 6th)*[189].

Code Constitution clash: There is a clash in Constitution between the code and metadata being created and held in BK (Linux kernel code), and that of BK's code. Accessibility and the ability to make changes was Stallman's main reason for creating the Free Software Foundation, and these rules are embedded in the GPL. What happens to code, or is allowed by the code, is embedded and dictated in the Constitution and thus affects what it can do and how it can do it. Code is thus law (Lessig 1999a). Code controls what machines do and Lessig's question is how do we in turn control code so that we control machines. Code controls, orders and organizes social and economic activity. Lessig's metaphor of code can be said to take a rather technological deterministic attitude towards open source and software in general, and yet we do see in this case study some real indication of this controlling nature of code, or material agency. Information and knowledge is power and so is code so whoever controls it and can access it, has power. A fear of losing control over their jointly created knowledge emphasizes the hybrid nature of Linux kernel metadata, which is nearly as valuable as the code itself. Without the metadata the repository will not be complete, and without the metadata the code will not make much sense, thus this repository will no longer be a learning repository. This is a breakdown threatening to happen, and indeed the above message precipitated a long discussion on this topic which did lead to a rather serious breakdown in the collective.

Hybridization

Open source distribution for BK: Laws are embedded within the code and laid out in the licence. Laws are unyielding but not entirely inflexible. We know that BK has managed to exploit some flexibility even with the GPL, because according to Machek (2002) BK contains some GPL'd

l does not contain pointer to the sources. [I
u should shut up and be glad you may use this
en, he tried to punish me for pointing at those

bitkeeper docs does not mention its GPL-ed, anc
pointed couple other issues.] Larry's attitude is "yc
for free" and "sue me to get GPL issues fixed". Th

0210.0/1918.html

189 http://www.uwsg.indiana.edu/hypermail/linux/kernel/0

mistakes by withdrawing installer from GPL. Nice attitude, I'd suggest you to stay away from bittrojan^Wbitkeeper[190]" (Machek, 2002 – Mar 18th)[191].

McVoy claimed that at first "*BitKeeper is an Open Source distributed revision control system*" (McVoy, 1999 – Feb 21st)[192]. This seems to imply that software can have different licences for various functions. The GPL obviously cannot take into account every contingency and there is often a loophole in the law that can be manipulated, thus it seems that no constitution is beyond circumvention. The GPL has never been defended in court but that should not imply that it is not enforceable (Moglen, 2001)[193]. However, Moglen (2001) believes that perhaps there should be a greater emphasis on enforcing at least a few GPL cases as that would make potential wrongdoers wary. This could also explain how so much has been allowed in BK's case.

Hybrid metadata: Proprietary software can have open source distribution and metadata created by proprietary software can be GPL'd. At least this is what McVoy claims. In late 2002 McVoy was asked "[is] *the commit messages and other BK metadata GPLed?*" (Anonymous developer, 2002)[194]. His reply was "*That isn't up to BitMover, that is up to Linus. Look at clause 3(b). Linus, as the creator and owner of that repository, can put anything he wants in BitKeeper/etc/REPO_LICENSE so long as it doesn't conflict with the BKL. He can say that all the metadata is GPLed if that's what the world wants. It's his issue, not ours. This clause is very*

ense, the BKL is automatically not granted.
ed to" (McVoy, 2002 – Oct 6th)[195]. But little
ely renege on his statement, "*the metadata*
e that metadata, you did not" (McVoy, 2003
come GPL'd, even claiming ownership. This
en the metadata, but as BK allowed for it to
't have existed in the first place if it hadn't
of it. Torvalds in 2002 didn't exercise the
metadata open to appropriation from BK
id and contestable nature and ownership of

3.2/0528.html
2.2/0812.html
0.0/1978.html
0.0/1978.html
21.html

access the repository. If you don't agree with his lic
He could use the BKL to enforce the GPL if he want
over five months later McVoy went on to complet
which makes BK work, that's ours, not yours. BK mad
– Mar 16th)[196]. And denied that the metadata could be
is in spite of the fact that neither he nor BK has writt
be written and stored, McVoy argues that it wouldn
been for BK. On this premise he claims ownership
control given to him and thus left the Linux kerne
owners. The collective is now left with a rather hybr
the kernel metadata.

[190] http://atrey.karlin.mff.cuni.cz/~pavel/bitkeeper.txt
[191] http://www.uwsg.indiana.edu/hypermail/linux/kernel/02
[192] http://www.uwsg.indiana.edu/hypermail/linux/kernel/99
[193] http://emoglen.law.columbia.edu/publications/lu-13.html
[194] http://www.uwsg.indiana.edu/hypermail/linux/kernel/02
[195] http://www.uwsg.indiana.edu/hypermail/linux/kernel/02
[196] http://www.ussg.iu.edu/hypermail/linux/kernel/0303.2/0

Ajar source: The nature of BK itself, surprisingly, is also not very clear. The licence is proprietary but even the actors that don't agree with using BK for kernel development only go so far as to call it "*ajar source*" (Miller, Sept 28th, 1999)[197] a neat name for it as another hybrid, neither clearly open or closed source. Of course McVoy insists that, "*BitKeeper is as open as it can be. You get source, you get to wack it and redistribute it for free*[198]*. The license isn't as open as you want but that's because we need to make money in order to support BK and move on to developing the next generation of tools: bug tracking, binary file revisioning, project management, etc.*" (McVoy, 1999 – Sept 27th)[199]. So the licence is not open yet McVoy claims BK is open. One possible interpretation of this paradox is that the licence is not the clear cut constitution of software and is indeed open to rife manipulation, or this could be indicative of the pragmatic approach always preferred by the actors of the collective. Whatever is able or demanded to keep the collective from breaking down, and holds it firmly together to create software will become permissible. The latter argument resonates more clearly from the case study, and if hybridization occurs, that is another attempt to ignore rules in order to sustain the collective.

Hybrid open source model: There is then some support for not only dual licensing but also a hybrid OS model, "*I'd like the Linux community to support Larry in working out the details for this "hybrid" model for "open software". I and many others would like to do something like what Larry is doing. In particular, it allows *small* software groups to produce very high quality software and still pay the mortgage and send the kids to school. Let's be pragmatic, except for over-caffeinated college students, the rest of us simply cannot afford to indulge in "free/open" software with out a clever way to make a few bucks to put food on the table. I'd like to propose a topic for Eric R. (our resident pundit and deep thinker): Software as a Cottage Industry*" (Leighton, 1998 – Oct 5th)[200]. Back in 1998 when this developer spoke of such a move towards hybridization there were not too many such attempts as yet so this idea was quite novel, even ahead of its time. Now of course we have other hybrid business models, licences, strategies and software. What is interesting is that there emerged a realization of the need for hybrids. The constitution may be fixed but patches and developers manage to ease the collective away from any extreme to some middle ground where the licenses, business models and constitutions can be more flexible and inviting, and more pragmatic.

[197] http://www.uwsg.iu.edu/hypermail/linux/kernel/9909.3/0550.html

[198] The developers will be able to learn from example and this contradicts the issues which emerged later when this collective stopped using BK and began its own VCS, GIT. It was rumoured this was forced because McVoy got annoyed at some attempts to reverse engineer BK. There would be no need to reverse engineer if the source code was available and visible.

[199] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9909.3/0558.html

[200] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0545.html

Breakdown

Most, if not all theories of learning and organizing understand that change often causes some form of breakdown and vice versa. Different authors call by it different names, double bind or paradox (Bateson, 1972), crisis (Weick, 1988), breakdown (Patriotta, 2003; 2004), and mismatch of outcome to expectation (Argyris and Schon, 1978). Breakdowns vary in their intensity. If a breakdown is managed well the outcome or change can reflect a better understanding of the problem, or learning. However, not all breakdowns are manageable and sometimes result in fatal fragmentation. Below we deal with a few different kinds of breakdowns that the Linux kernel collective suffered and how they overcame them.

Fear of Breakdown

Loss of critical mass: The threat of a breakdown is feared by the Linux kernel collective. A fork in the collective, though allowed, is considered to be a taboo act and only to be undertaken when there is no alternative left. But the more experienced developers do admit that the ability to be able to fork a project and go your own way with your code base is a safety valve. When the pressure becomes too much and a serious breakdown occurs then the best way forward is to take the code base and develop it in any direction you want rather than stay in a situation where the code suffers. A fork can also lead to greater chances of improvisation and learning within the collective. However, there is also some real wisdom in discouraging forks in OS projects because such a step dilutes the developer and knowledge base, and all segments of the project become in danger of losing critical mass. "*And you think it's a good idea to have the linux community divide*

[redacted]

> *no longer adequate to meet the challenges of success, and some way to institutionalize and distribute the leader's role has to be found. Movements that fail to make this transition die, generally by implosion or fragmenting into feuding sub-sects. If you were familiar with the historical precedents, Rob, you would understand that your modest proposal*[202] *re-enacts a common pattern. A relatively junior member of the movement, one with few political ties, sees the developing stress fractures in the organization of the movement and proposes a modest, incremental change to relieve some of them. Conservatives interpret the attempt to separate and institutionalize part of the founder's role as an attack on the authority of the founder. Huge flamewars ensue, with the original pragmatic sense of the proposal often being lost as it becomes a political football in the movement's internal status games. Sometimes the first such attempt at institutionization succeeds. More often, the movement has to go through a series of escalating crises (burning up would-be reformers each time) before anyone finally succeeds in changing the movement's internal culture. Religions go through this. Secular social movements go through this. Companies founded by brilliant entrepreneurs go through this (the B-schools have a whole literature on "entrepreneurial overcontrol" and its consequences). It's one of the dramas that gets perpetually re-enacted; it's built in to our wiring. The unhappy truth is that even *successful* transitions of this kind are invariably painful, and often leave deep scars on the survivors and on the institution that arises from the transition*" (Raymond, 2002 – Jan 29th)[203].

The interesting point he makes is his understanding that this crisis stems from a desire for *over control*. The founder, in this case, Torvalds, does have a strong need to control every aspect of his project. He is beginning to micromanage when the reverse is needed from him because the project is far to large for him and just a handful of people to control. The collective is undergoing some drastic changes and learning to cope with this change is not straightforward. Each breakdown the collective faces either makes it stronger, or may break it apart, but no matter what the result the only constant is change. In 2002 control of the collective, the code and coordination is slipping away from Torvalds and he is reacting badly.

Sanctioned knowledge: Vger [CVS] often held 'more' information and patches than did the tree kept up-to-date by Torvalds, "*We're painfully aware of this. We do sync up with you though, so the vger tree does have what he needs along with your changes*" (Dougan, 1998 – Sept 28th)[204]. VCS holds knowledge that Torvalds has not sanctioned and has not even read. The VCS seems more informed than Torvalds and this could be what bothers Torvalds about CVS in particular. VCS is gaining more control and power than the leader. Of course control is needed but it is how this control is organized and managed that is important. Control allows for organization and thus

[202] See http://www.uwsg.indiana.edu/hypermail/linux/kernel/0201.3/1000.html for more details on Rob Landley's Patch Penguin Proposal

[203] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0201.3/1393.html

[204] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0548.html

better access to information and knowledge. An interesting question to debate would be that by bringing in a moderator for the LKML would we get a more organized approach or a more controlled one?

Breakdown in Collective

A breakdown for Torvalds soon becomes a large concern for the collective because his inability to scale is affecting the entire development process. The breakdown in communication between Torvalds and the collective now threatens a much larger breakdown. A lack of VCS may have caused this but at the same time we see that not any old VCS is acceptable to the collective, especially the leader, who will not settle for CVS. See Miller (1998 – Sept 29th)[205].

CVS is not accepted by Torvalds but some software is needed. This crisis may eventually lead to some VCS being adopted, but there will have to be a VCS very soon. The OS community does not rise to the occasion and create something that fits Linux, instead we see proprietary software begin to insinuate its way into the open source collective. The leader feels threatened by CVS because too much of his control is slipping away from him, which also means that the collective and his project is beginning to desert him.

Collective breakdown: Not only are individuals facing breakdowns, so is the collective and all this because there is a breakdown in the patch submission system. The patches are in revolt and are using their developers as spokesmen to voice their problems, "*This just because it slows down the process, and is a lot of pressure when things are tight. Receiving a large number of patch all at once, or a single huge patch is daunting even when you are blessed with copious spare time. But when you are working a regular job, taking care of family, and under the gun to release a kernel that senior folks suggested could be out a month ago (admittedly VERY tentatively), well, you might be more than testy. I think Linus's explanation of how he wants to work is reasonable. I'd bet he gets 500 mails a day, many of them patches to this and that. He probably pays attention to the ones from the senior people first, but I bet he does a wholesale slash and burn after a certain point, just because it's too much to deal with at once*" (Todd, 1998 – Oct 2nd)[206]. Todd provides an alarming visual picture of hundreds of patches, instead of becoming part of the kernel being deleted into oblivion. So much hard work is wasted and no collective can afford such squander. If patches are not accepted then the Linux kernel will not grow, the centre of calculation (software) will not grow and change and thus stagnation will set in which will be the end of the kernel.

[205] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0732.html
[206] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9810.0/0243.html

Breakdown in Learning

Sub-optimal coordination: An historical archive was called for to protect Linux kernel code for the future. This issue was brought up at a time when the collective was worried about the future of Linux as Torvalds was facing scalability problems and patches were being dropped, around 1998. The patches that were dropped by Torvalds are already lost knowledge, but the collective doesn't want to lose old versions which are scattered because there is no one recognized repository for kernel code. The other problem the collective faces is that without one recognized repository or version of Linux kernel code, and vger is not acknowledged by Torvalds, some parts of the collective are working to a different version and others are unaware that some problem has already been solved and wastefully continue to look for a solution. This is a misuse of resources and one of the basic tenets of OS is reuse as opposed to trying to reinvent the wheel, "*Good programmers know what to write. Great ones know what to rewrite (and reuse)*" (Raymond, 1999). The collective soon recognize that "*there is something going wrong with vger <-> Linus relationship. 2.1.123p2 was out and a lot of people complained about the fbcon.c breakage. It was said, that there are patches in vger which cure this, but 2.1.123 still suffers the same problems. So the coordination between vger and Linus is suboptimal*" (Garloff, 1998 – Oct 2nd)[207]. There is a loss of knowledge as software patches are being ignored and though solutions to problems exist some parts of the collective are oblivious to them. There is learning but it is not being passed around so the circulating reference has been broken in a way that it no longer moves and takes on new ideas to the actors that it comes into contact. Put more strongly the Linux kernel release no represents the collective – a situation that cries out for some software to step in and solve the problem.

Disorganization: Too many breakdowns lead to disorganization, and the potential learning involved cannot be assimilated, "*It is really a pity that the most popular free UNIX is so chaotic, and it's only becoming worse. Please don't misunderstand me: I'm not trying to persuade anyone* of patches and bug fixes but as this is not assimilated back into the collective – waste of resources and fragmented learning.

Breakdown in patch queue system: The Linux kernel collective were faced with a leader who openly stated that "*if some person cannot be bothered to re-submit, I don't WANT the patch. Anybody who is not willing to take that much care of his patches that he can't maintain it while I haven't accepted it, I don't want to accept patches from anyway*" (Torvalds, 1998 – Sept 29th)[209]. Developers were irritated by this attitude because the larger problem in this breakdown was not that they were not willing to take responsibility for updating their patches but that developers were forced to work on solving problems for which fixes already existed. Miller's (1998) response to Torvalds reflects how some technological aid is clearly needed to overcome the invisibility of the patch queue system, "*Linus, the problem is that when I submit a patch, I don't want to unnecessarily overload you by resubmitting on too short a notice. So I wait, and wait. Nothing happens. People are ignoring the fact that we had solved this problem (at least in my*

Technology Makes the Collective Durable

Linus does not scale: Jitterbug was discussed by Miller (see above) as a potential solution to the problem faced by Torvalds and the Linux collective, as were so many other tools like Aegis, SubVersion, PRCS, and a patch robot like LSM Robot. However, CVS had the strongest backing and network. Of all the tools available CVS was the most advanced and used by many OS projects for coordination like BSD and Mozilla but the Linux kernel collective didn't adopt it "*because Linus doesn't like CVS. CVS is [IMHO] exactly how the Linux kernel 2.1.x stuff should be run simply because Linus doesn't (and now by his own admission doesn't) scale. Linus does have some good reasons for not liking CVS in that CVS has no real control by 'area'. It lacks the ability to easily say things like xyz has remote cvs access to drivers/sound/*, and to include/linux/sound.h but if it's the latter notify me... It works, nothing else free comes close. Right now for example if you want to know why a patch was added to the Linux kernel you have to hope its in my patch archive or it was added via vger and thus notified. Yes the current system is a complete mess. Yes it needs fixing, but someone has to find a system that works and Linus is happy with. Jitterbug was close for tracking, but doesn't solve all the scaling mess. If you have the solution then do the yelling. Also if someone has a good way of efficiently replicating a 40Mb CVS tree over a modem to an anonymous cvs server let me know. Then I can look at putting 2.0.37pre1 or whatever under CVS*" (Cox, 1998 – Oct 3rd)[211]. The collective is agreed on the idea that some technological aid is necessary in order to sustain Linux. This draws on Latour's understanding of the role a machine plays in a collective, "A *machine*, as its name implies, is first of all, a machination, a stratagem, a kind of cunning, where borrowed forces keep one another in check so that none can fly apart from the group" (1987, pp128-129). VCS is potentially a machine or machination and the collective understand that in order to sustain their collective or assemblage they need one, and soon. VCS has convinced them that, though there are a few other solutions that could be adopted like a patch robot, or just another way of organizing through emails or a webpage, the collective *must* use a VCS because that is the *only* way Linux kernel will not fork or break apart.

Again we see the pragmatic approach taken by the collective. Code is important and the continuation of the Linux project is crucial, so anything that is required to achieve that end will be acceptable. There is a communication and coordination breakdown and CVS, especially the vger version is offered as a remedy, at least a short-term one. A VCS might give the entire collective access and transparency so that knowledge [patches] is visible to all. Knowledge is power and brings control to who holds it. However, CVS does not seem to provide the kind of accountability that Torvalds desires. Perhaps CVS provides too much transparency, and transparency to *all* and

[211] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0347.html

that is why Torvalds is not keen to adopt it. We thus understand that perhaps, depending on who in the collective is asked this question, transparency is not always needed or even appreciated. Each VCS provides varying degrees of transparency.

Conclusion

Three main points that emerge from the 'sustaining the assemblage' theme include:

Code sustains a collective: Interest in the future of code brings various actors together and it a similar continued interest that sustains the collective. Code is both the software being created and other code like version control software that provides the infrastructure for software development.

Hybridization of collective, process, code, licenses...: Linux development demonstrates a clear move towards hybridization. Software with different licences mix together and either a blind eye is turned or such mixing is written into the licence. The collective is not just human actors or technological, but a complex intertwined mix of both. Non-humans contrive to bring other actors to assemble around and form a network. However, what is more interesting and of use to us is the idea "that we don't assemble because we agree, look alike, feel good, are socially compatible, wish to fuse together, but because we are brought by divisive matters of concern into some neutral, isolated place in order to come to some sort of provisional makeshift (dis)agreement" (Latour and Weibel 2005). This is what we discuss later as a creative space.

Technology balances learning with organizing: Technology and in particular VCS is responsible for maintaining a balance between learning and organizing. Weick makes clear that slack is a prerequisite for learning but too much of the former causes disorganization. Our study is indicative of how VCS is keen to both organize the actors of the collective but very importantly, to also ensure growth and learning.

Machine Agency: VCS, a Controlling Device

The third theme of this Chapter is that of material agency and how version control software is a *controlling* actor. Three memos of agency, control and ownership contribute to a better understanding of how version control software has the ability to act and make other actors do its bidding. It has agency and uses it to get control over the collective because it wants to own the software produced.

The *agency memo* expands on how action by material actors can be understood only when we consider the agency they have to respond, create and learn. VCS controls other actors or is used by some strong actor to control the larger collective in order to ensure the sustainability of the collective and the learning it has created.

The *control memo* shows how space as infrastructure, or as in this case VCS space, directly impacts on access and transparency allowed to the collective to manoeuvre. We also get a glimpse of the struggle between Torvalds and CVS for control of the collective but we eventually understand this to be the collective's attempt at regaining control over learning and thus its future.

The *ownership memo* makes explicit how the one who controls the collective becomes the 'owner' of the learning created and that is why Torvalds battles the VCS until he manages to find a VCS which is willing to become his apprentice rather than his boss.

Agency

The concept of hybrid agency which is inclusive of both human and material agency emerges here. The constructive-realist perspective of actor network theory provides a way to discover such agency as it emerges. This section considers two main areas, *agency and control* and *agency and learning*. The key ideas which become apparent when the link between *agency and control* is studied are; the dance of agency; material agency voicing its opinion through action; material agency controlling the ability of other actors to behave in the way they desire; the appropriation of technology by actors to control others; and control through transparency of agency. This section clarifies that technology acts on its own behalf or of that of another actor to control the behaviour of the collective. This is only made possible by the ability of both human and material actors to influence and this in turn implies a level of agency for every actor.

Discussion then moves to agency and learning. *Agency and learning* are coupled because it becomes apparent that actors are only able to learn because they have the capacity to act and need to learn in order to survive. The main arguments raised by this study connecting learning and agency are that; the able VCS is the one that is better able to learn; technology is forced to reckon with unlearning; VCS are able to detect difference; VCS can discover corruption; and can control time and space to search for learning. The ability to perform these acts reflects the capacity to learn in technical actors.

Agency and Control

Dance of agency: The role of version control software in open source development has changed and evolved over time. It provides a useful mechanism for coordinating work and workers but as we have seen through the Linux kernel study, it is more than just a simple tool. Technology and all material actors are capable and empowered to act because they have agency. Agency does not imply intentionality but it does entail the ability to act, react, and learn. This is a constructive-realist perspective on actor agency. We speak of ability but it is coupled with empowerment. Technology, just as other actors, must be empowered by others to operationalize its latent agency[212] e.g. every actor claims a network. The Linux kernel reality is then co-constructed by all kinds of actors, material and human. We have many examples of human and material actors working side by side to create and sustain the collective and the evolution of software. An expressive example of constructive realism is the relationship between the user of any VCS, in this case BK, and technology, "*If during any of these steps Linus changes any of these files, I bk pull, and with luck, bk does the nasty bits for me, and fires up the conflict resolution tool if needbe*" (Jones, 2002 – Mar 8th)[213]. The 'dance of agency' (Pickering, 1993; 1995) is well

ce to play a part in
oes exactly what is
o resolve the issue.
al material agency

issue is to resort to some material device. However, once they allow this devi
the unfolding of the problem it begins, gradually, to take control. At first it d
asked of it but each situation it is confronted with varies so it then improvises t
It is at the point where it begins to make its own decisions that we see re

potentially good code isn't doing a good job of defending itself." (Hand, 1998 – Oct 4th)[214]. Good code can speak for itself and does. The ability of code to speak through its smooth and efficient operation is indicative of agency in technology. If code is written well it is able to satisfy the need it was intended for and allow more efficient coding strategies and trajectories in the future which build on prior work.

Technology hinders: Material actors, because they possess agency, can also become an obstacle to others in the collective. Technology can be intractable "*Unix gets in the way less, whereas with VMS I found myself battling the API more to force it to do what I wanted*" (Gooch, 2002 – Mar 11th)[215]. It hinders as well as aids. It is controlling because it is able to apply force on other actors and deflect their decisions. Gooch compares various technologies in relation to how controllable they are which clearly indicates that technology is not the pliant tool considered by many.

Technology shows partiality: If actors share a similar ambition then, be they material or human, we see them collude to control others to achieve their purpose. The collective is sustained through

Deceptive transparency: VCS and other such technology bring visibility and transparency to the process of development. However, we are only able to see what technology 'allows' us to see. Thus we have a mixture of control and transparency or rather control through a particular type of transparency, a deceptive transparency. With the availability and openness of the source code in open source software the level of transparency is enhanced. The collective is able to 'see' more clearly what is happening, where, when and how, "*With a management system in place, a self-documenting system of changes would provide a method of viewing these sort of changes in a patch-by-patch basis*" (Mendelson, 1999 – Sept 27th)[218]. If open source is more transparent it lends itself more to reversibility rather than irreversibility. Mol and Law (1994) consider the idea of irreversibility in ANT and through the example of fluids and anaemia, explain how ideas can be mutable rather than immutable. Anaemia means different things to different people, even doctors, across the world. The idea thus mutates and is never the same from one actor to another. The idea or artefact in open source, because it is visible, lends itself to greater mutability. Each

jency and Learning

ood functionality reflects learning: Technology is able to mutate and evolve. In this section we
nsider the various abilities of technology that indicate learning. Material actors must have a
ice in order to be able to reflect their learning techniques and ability. As we noted above
:hnology speaks through spokesmen but also through effective and elegant action, "*Yelling at
ople will never convince them that BitKeeper is good. Only BitKeeper itself can do that*"
anza, 1999 – Sept 28th)[219]. A spokesman alone is not always enough to convince other actors. In
s example the VCS needs to convince the collective of its worthiness to be adopted by all
embers through active and efficient control and coordination. This takes the control and agency
ay from the core developers of BK once they release BK code. The developers may be unable
sustain themselves as obligatory passage points and BK, if accepted by the collective becomes
new and stronger OPP. Efficient technology is indicative of embedded learning. BK code
ıst be elegant and intelligent in order for BK to run smoothly and thus persuade the collective to
propriate it.

learning difficulties: VCS demonstrates agency and learning when it can *learn more but also
en it faces difficulties unlearning*. The spokesmen of the collective often speak of technology
d refer to how much a specific technology 'knows'. This could possibly be the usual
thropomorphizing that developers are often blamed with, but developers are also the actors that
to mould the technology in a particular way. It is very probable that developers and other

actors that interact very closely with software have a better understanding of the forceful nature of technology, "*Note that it's a lot easier to expand tools that already know about revision control than it is to expand tools that have never known about it*" (Torvalds, 2002 – Mar 15th)[220]. Past learning can help with future learning. This is true of both human and material actors. Nevertheless, technology is often intractable. And like human actors VCS tends to retain embedded learning which can make it reluctant to try a different approach. How it is structured determines the possible trajectories future learning can form. Material actors, like human ones are just as prone to the difficulties of unlearning and unlearning is the most difficult part of learning something new.

Detecting difference: VCS exhibits agency and learning when it is able to *detect difference*. VCS is able to decipher differences in software, no matter how minute, thus it appears to be intelligent, "*The push command is used to update your parent repository with your work. The push command will never create conflicts in the parent repository, instead it first does an implicit pull to make*

changes" (McVoy, 1999 – Sept 27th). It takes charge of activities, it controls and organizes the actors in the collective. It commands, patrols and reminds, perhaps even chides with reminders. Through these various mechanisms of organizing VCS makes sure that nothing precious is lost. The ability to make comparisons between various patches, and in the case of BK, to then make a decision autonomous of humans via the automerge is strongly indicative that technology has the ability to make decisions based on understanding.

Controls time and space: VCS illustrates agency and learning when it *controls time and space*. VCS folds space and brings it together at any point in time. It has the affordance and agency to be other VCS. It thus has *more agency* than most VCS because it does control corruption and control time and space.

Control

The second memo of control gave rise to issues of space and control, human versus material control and finally, collective control. The section on *space and control* elucidates how controlling space brings control through transparency because control over space is possible only through access and visibility. The second argument this section makes is that the underlying infrastructure and architecture of any space dictates how fully control can be exercised over other actors.

Human versus material control helps to explain how powerful material agency can be because we see Torvalds struggling against it to regain control over the Linux collective. VCS is aware that it has the ability to be more efficient than Torvalds so it attempts to bring this strength into focus to prove how it is more worthy to control the collective. However, we also see how Torvalds manages to manipulate the VCS of his choice to become his apprentice OPP.

Finally, *collective control* stresses how the assemblage is held together through collective control and continual pressure. The need for a VCS in the collective emerged from a realization that collective control was only possible with some material actors that would provide better levels of access to all actors. Testing or 'eyeballing the code' written by other actors is another form of collective control in open source. And finally collective control comes about through the ability to read the code of everyone. This is past learning which is used by all actors to trigger new learning (code) and is a necessary prerequisite for the collective to sustain itself.

Space and Control

Control of space: Controlling space gives an actor immense power through the transparency it affords. The control exercised by Torvalds over the patches and developers of Linux is useful because as Torvalds points out he can then step in and fix issues as he is able to examine the overall structure and direction of the project as he says in his first major comment on VCS (CVS) in 1995, *"I'd be more than happy to make some easy change logging facility available, but I'm afraid that I don't like the idea of having developers do their own updates in my kernel source tree. I know that's how others do it, and maybe I'm paranoid, but there really aren't that many people that I trust enough to give write permissions to the kernel tree.. Even people I have worked with for a long time I want to have the option of looking through their patches _first_, and maybe commenting on them (and I do reject patches from people). Also, the fact that all the patches go* *through me also means that I generally have a reasonably good idea on what people are working on, so when problems crop up, I also have a chance in hell of getting them fixed. Or I can decide (unilaterally) to revert a patch that results in problems. For once, Linus being lazy is actually only a secondary concern*" (Torvalds, 1995 – Dec 11th)[223]. The maintainers only have control over one specific module and even then part of their responsibility lies in reporting back to Torvalds and always keeping him in the loop. VCS might provide Torvalds with the panopticon position of being able to see and control all and everything – a very powerful position. But CVS threatened this view for Torvalds because it did not allow control over space (as BK later managed) and thus we see Torvalds fight hard to retain his stronghold. He dissuaded actors from adopting CVS by clearly refusing to embrace it himself. How space is organized or created around actors is important because the space and architecture brings with it real power and control.

Human versus Material Control

Torvalds' tussle with VCS: The dictator side of Torvalds is very apparent and intense on occasion. His reluctance to adopt CVS for kernel development reveals less of the benevolent and more of the dictator side of his character. He struggled for control with CVS to regain dominion of *his* project "*Note that saying "it's in vger so you're wasting your time" is still completely and utterly stupid. The fact that it is in vger has absolutely no bearing, especially as there's a lot of stuff in vger that will probably never make it into 2.2*" (Torvalds, 1998 – Sept 28th)[224]. And even "*I have stopped synching up to vger a long time ago. Anybody who still thinks vger has any relevance to the standard kernel is very much misguided*" (Torvalds, 1998 – Sept 28th)[225]. Torvalds has final say over what code makes it into the Linux kernel release and he intends to maintain that decision-making power. Struggle between Torvalds and CVS over who will control the code continues. Control means power to make decisions and dictate to others, "*The reason I'm disappointed is that vger in particular has been acting as a "buffer" between me and bug-fixes, so that now we're in the situation that there are obviously bugs, and there are obviously bug-fixes, but I don't see it as such, I only see this humongous patch. I don't know what it fixes, because vger has kept me out of the loop, and quite frankly I don't have the time to look at several hundred kilobytes of compressed patches by hand. And I refuse to apply patches that I don't feel comfortable with. As a result, I want people to tell me what the patch fixes. And that probably means by now that I need to get each driver update on its own, with explanations. This is exactly the same thing that made me hate vger when it came to networking patches. And I'm going to ask David once again to just shut vger down, because these problems keep on happening*" (Torvalds,

[223] http://www.ussg.iu.edu/hypermail/linux/kernel/9602/1096.html
[224] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0554.html
[225] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0540.html

1998 – Sept 29th)[226]. Torvalds wants only one dictator in Linux, he is not willing to share control with CVS because this opens the flood gates to the collective for joint control. Democratic as many open source projects are promoted to be, this example of a tussle between Torvalds and CVS for supremacy indicates that the benevolent dictatorship model has more dictator in it than often credited.

Control through efficiency: But if Torvalds needs to strive to regain control it proves just how much influence VCS is able to wield. VCS voices its concerns and even flexes its' muscles through various spokesmen, "*But sorry, there are driver updates for platforms you don't give a crap about (i.e. non-Intel) and this is where the immense size comes from. Martin and Geert are quite diligent about fixing all bugs reported to them, and now that it has all accumulated you still are not taking the patch. What do you want him to do? It is at a state where you can't just "pick out 200 lines of the fixes" because it simply is a sizable patch. Apply it and be done with it, then you can be sure any further changes will be tiny. Currently it is at a stale-mate. People are maintaining this code, and if you block patches they can't do their job. I ask you to be a part of the solution instead of the problem*" (Miller, 1998 – Sept 29th)[227]. VCS' strength is efficiency of control which Torvalds is unable to match. Open source democracy is at play because even the leader is not above being scolded or told to step aside when the survival of software is at stake, and various software, both VCS and kernel patches are working together to add pressure to the situation. Software has the most to gain from a more efficient coordination process, be it VCS software or kernel patches. So many lines of code are not scrutinized and thus frustrate the developers because it becomes impossible to debug. The aim of a patch always remains the same, to be included into a final version of software so that it has a purpose and is recognized.

Technological successor: Another instance of Torvalds battling technology is when the question of naming a successor was raised, "*it's a sign of a healthy organization to have folks that could step up hanging around being groomed. It means that the organization is more important than the individual and I think that is the case here, with no disrrespect intended*" (McVoy, 1998 – Oct 2nd)[228]. Torvalds was not inclined towards naming a successor. The collective felt the need for an heir so that the kernel assemblage could be sustained, "*Any leader needs to delegate. Grooming a replacement, and keeping said replacement on board as a hot spare, could be done by having Linus delegate some authority to this person. He already has people he more or less trusts to submit good patches... I agree that the grooming process would have to be preceded by (or*

[226] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0766.html
[227] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0672.html
[228] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9810.0/0258.html

include under its umbrella) a selection/weeding-out process...." (Kwrohrer, 1998 – Oct 2nd)[229]. This message was written in 1998 and now it is 2006 and still there is no mention of any replacement by Torvalds. It can be argued that his chosen successor (at least for a time) was not a human, but rather it was a VCS – BitKeeper. It is interesting to note that Torvalds no longer uses BK but has decided to create his own VCS, GIT. It is rumoured that this step was forced upon Torvalds by McVoy because a kernel member (Andrew Tridgell) was accused of trying to reverse engineer BK, but perhaps BK was also beginning to gain too much control over kernel development.

Collective Control

Assemblage crumbling: There is a struggle within the kernel collective between human and material actors for greater control but the overall drive is towards collective (hybrid) control. The collective is stronger when it acts as a uniform assemblage, rather like the Borg (Star Trek). We see Torvalds fight CVS for control but the collective was not entirely happy. This dissatisfaction stemmed more from his continued ignoring of patches and developers, but it became focused on a Torvalds versus CVS argument. Torvalds felt that he was losing his collective nearly caused irreparable damage to his status as benevolent leader. The ironic point in defence of his own leadership was that by refusing to loosen control over the collective he was in more danger of losing the collective and indeed causing it to fracture, "*Linus You do realize that your powertrip lately is really hurting Linux development. Linux has been getting a bunch of good press lately (finally) but if this ego/head-in-sand/etc trend continues.. 2.2 will never go out the door and we'll have this buggy heap called 2.1 still around. 2.1 has been drawn out too long and people are bailing ship left and right over stupid arguments. Jitterbug flopped.. vger is now flopping.. you've already lost some damn good IDE coders, and I'm sure you're going to lose more at this rate. I hate writing a message sounding like this but I think it should be said*" (Tiensivu, 1998 – Sept 29th)[230]. The clashes and problems that arose between vger and Torvalds threatened to dissuade the collective from holding together. They are together because they offer code and constant feedback to each other, but if the code and feedback is ignored, they no longer feel needed. This is a breakdown of the assemblage.

Micro-management of code: There are various levels of control and different VCS are able to provide a choice of granularity of control to the collective. The developers know that they need some technological help to coordinate their work and keep the assemblage together yet the help mustn't be at the expense of giving up too much control, "*CVS is ugly anyway, because there's no fine-grained access control (i.e. either you can change everything at will or you have only read-*

[229] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9810.0/0260.html
[230] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0789.html

only access). I really hope the PRCS people would do something about their planned client-server support, it looks just like what we'd need -- I have the whole kernel under PRCS locally (every release since 1.0.0) and might be able to share the thing somehow, once the PRCS support is there" (Urlichs, 1998 – Oct 2nd)[231]. Fine grained access control is required, a need for filters in the VCS that bring some flexibility to Torvalds. Torvalds is better able to delegate work and organize his collective if what organizes the code, and thus the workers too, operates in the way that he wants. He wants the VCS to answer to him and not the other way around. But does the code care who it answers too? Perhaps not, because the code is worried about survival, meaning it wants to make it into the final version release of Linux kernel, and it doesn't matter how or who does that for it. The collective is eager for patches to proliferate and become part of the Linux kernel, however, it must grow in quality and strength, so the filter in the VCS must ensure that only the fittest of all patches survive.

Democratic collective control: Collective control is also manifested in testing or 'eyeballing the

collective. We could also see this as another device or machination to hold the assemblage
together. This form of cont ol is made possible by openness and transparency in both the process
and code. The email extract below reflects this and a number of other interesting points,

> *"Hey, where can we get a hold of these people? :-) Seriously, I can't*
> *imagine mai taining a project like the Linux kernel with only one*
> *man committi ng to the CVS tree (if any). The way to go is to have*
> *several peop e committing, but make the CVS tree so (readonly)*
> *accessible tha t everyone can review old-vs-new a few hours after the*
> *commit. Even the BSD projects with their smaller people base have*
> *enough peop e to ensure quality that way. I'm sure the masses of*
> *kernel-aware Linux people will be a better (more scalable) substitute*
> *for judging o ver (committed) patches than a single person can be.*
> *The most imp ortant thing is probably to look for special cases the*
> *new code wil break and that the committer haven't been aware of. A*
> *single person with a wide overview may be good in this area, but it is*
> *not scalable. Also, a patch that was committed, but backed out after*
> *protests may till be of value, since it:*
> - *Docu ments that this way doesn't work (and why), better than*
>   *a mai ling list archive can.*
>
> *Personally, I climb up the walls everytime I look at a Linux kernel*
> *patch becaus I don't have committ messages for the diffs. They're*
> **so* handy"* (Cracauer, 1998 – Sept 30th)[232].

In this extract we see the ne ed for democratic control over the collective voiced clearly. Cracauer
speaks of having more than just one or a few people given commit rights to the kernel but many
read-only. This in turn lea s to greater testing of the code and thus better quality. So we have

[231] http://www.uwsg.indiana.ed u/hypermail/linux/kernel/9810.0/0196.html
[232] http://www.uwsg.iu.edu/hyp ermail/linux/kernel/9809.3/0898.html

collective control (transparency) to guarantee that no one takes undue advantage, and we also have the significant argument of collective control through eyeballing of the code to certify quality. Another point made apparent in this email is how VCS makes possible learning and knowledge accumulation i.e. as a learning repository for this collective. The collective controls what is considered to be quality code, but the examples of code that don't inspire more work are also made available through the VCS (through the trajectories that could be followed) to provide a loop of learning to both new and old developers.

This developer also speaks of the limited information which is given along with the patches in Linux. He claims that this is irksome to the collective because the code alone cannot 'talk' to other actors. There is a need for commit messages or metadata [in BK] to make the patches understandable. Without the commit messages the developers find it hard to manipulate the code. Inadequate information thus leads to less control for the collective.

Ownership

The final memo that contributed to the theme of material agency is ownership. Ownership ideas gave rise to an understanding that the collective is eager to regain/retain collective ownership, and that collective ownership leads to collective learning.

Regaining collective ownership indicates how the collective began to feel incapacitated by Torvalds' strong sense of ownership of the Linux kernel collective. The campaign to ease it back into their own power was manifested firstly into a drive to make the collective aware that some form of help was needed to control Linux, and secondly, by brainstorming for ideas of what form that help should actually take, with a clear suggestion that VCS should be used. *Collective ownership for collective learning* makes it apparent that the collective needs to own what it creates because this is the reason for its being. If it owns the software produced then the collective is guaranteed that Linux will continue to evolve and learn and thus in turn it too will prolong its existence.

Regaining Collective Ownership

Torvalds initiated the project but he does not own all of it, however some actors of the collective feel he is the saviour so "*the simplest way to reduce pressure is to accept just the 'way he wants it' and to follow it. (don't forget, he's the copyright holder anyway) Given the enormous Linux users pool (estimated 8 million?), there ever will be a lot of 'patch noise' and Linus can't apply it all anyway. If we now try to 'administer' the noise in order to not loose anything, even a very clever strategy cannot avoid, that the queue accumulates to death. On the other side, the way Linus* *handles patch input will lead to sort of 'Darwinisme' such that only the 'strongest' (re-posted often) demands for inclusion will succeed. This 'filters' by itself. So, just accept Linus' way to handle it. As he often says (and is right saying it): the simple solutions are the most stable (and effective) ones*" (Lermen, 1998 – Oct 2nd)[233]. Torvalds does not single-handedly own the kernel. In fact under the GPL (Constitution) the form of collective ownership is such that each contributor owns that small section or patch s/he submitted and nothing else. The reason why it is called collective ownership is because in order for someone to download GNU/Linux the numerous authors or owners [of their patches only] give mutual consent to the third person to take their work and use it. Contributing a small patch to GPL'ed software implies your consent to allow others to use your patch again and again. Each member of the collective does not own every patch of Linux, the member only owns what he or she created. In spite of this indirect form of 'collective ownership' we see that Torvalds persists in calling Linux 'his project', his repository, his working area and his tree. Further, he also expects preferential treatment based on this

"If a person can't be bothered, as the maintainer of a piece of software, to respond to my submissions, maybe I don't won't want to work on that piece of software. This all brings back horrible remembrances of an altercation the gcc developers had, what was the result? EGCS. So if Linus keeps on acting like a Richard Kenner, then fine, this will be the result, the developers will go elsewhere and address the development process problems Linus refuses to install permanent fixes for. We had it going so well too, using Jitterbug. It was the answer, or at worst it made things better, and now it's off for the moment and we have the same problem again. Maybe we have to come to terms with the fact that it is possible for a projects size to just require that there is some mechanism to "mechanize" the development process. This can come in two forms:
1) Something like Jitterbug
2) Letting more than 1 person be the only ones who can make direct changes to the source
#2 is not an option to Linus, we did #1 and it worked, now #1 is gone and we have problems again. I didn't like when Jitterbug was disabled for code-freeze, but I understood the decision and kept my

[redacted]

Sept 30th)[237]. We hear the technology call out softly at this point. It wants to become a member of the network. And the collective is keen to find any way to ease some control back from Torvalds. They want ownership of their code because this ensures survival of code (learning). It is precious and must be protected and conserved. Part of the building, learning and conservation process is testing. In OS this is predicated on Linus' Law of 'given enough eyeballs, all bugs are shallow'.

Broken feedback loop: Eric Raymond, in a long monologue below, expressed his concern with the broken process of development that Linux suffers from without any actor like VCS helping. His main worry was that the feedback loop (deutero learning) is damaged and this will dissuade developers from creating more learning and this will in turn adversely affect collective learning.

> "*Linus is god until *he* says otherwise. The basic bazaar strategy still works (the flakiest 2.1.x I've ever seen would be considered production-quality by most OS vendors) but the ad-hoc way integration is presently handled is foredoomed to increase frustrations on all sides. Patches get lost. Patches get dropped. Patches get missed. This is bad and wasteful in itself, but it has a secondary effect that is worse -- it degrades the feedback loop that makes the whole process work. Anybody who starts to believe they're throwing good work down a rathole will be *gone*. If that happens too many times, we're history. These risks are bound to get worse over time because both system complexity and the developer pool are increasing.*" (Raymond, 1998 – Sept 30th)[238].

Raymond is an enrolled actor who believes that Torvalds is the absolute OPP without which Linux as an evolving collective won't survive. Enrolment in the case of Torvalds' followers implies that they accept him as owner and 'god'. God in this sense is an actor who controls all, sets the agenda, determines the direction that Linux will take and constrains the ability of other actors to act when their actions are in conflict with his own. The more compelling concern for the collective seems to be that their OPP might breakdown, and though that doesn't seem to concern Torvalds overmuch. It lays bare something particular about this collective. The collective is worried about the project and see Torvalds both as the problem and salvation. They don't act out of pure admiration for Torvalds but rather from a need for the survival of Linux software and this is what compels the collective to follow him. The real OPP is the Linux kernel and the patches that circulate and become centres of calculation for the project. They are the real owners of Linux. Torvalds has a role to play in holding the project together, but it is the patches created by the developers that circulate the collective to ensure the survival of Linux

[237] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0987.html
[238] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0849.html

Access to metadata provides collective transparency: Finally, the collective wants ownership of not just the patches but all the annotations that are added to the code. It is such annotation, metadata that makes the code understandable to the collective at large. Thus when the metadata held by BK was threatened by McVoy the collective panicked about the survival of Linux kernel development in the future, "*As for the data, you are right, we don't own that. As for the metadata which makes BK work, that's ours, not yours. BK made that metadata, you did not. If you don't like those terms, convince Linus and friends to get off of BK. That would be just fine with us*" (McVoy, 2003 – Mar 16th)[239]. Information is power. Learning is the opportunity to gain more power and control. VCS makes learning possible and creates a collective within a collective. The collective that obeys the VCS is given access to prized information.

Conclusion

The three main ideas to emerge from this narrative of machine agency include:

Fluid agency: Agency is not contained within anyone particular type of actor, it is a becoming that passes back and forth in a network which changes form and becomes stronger or otherwise depending on who attempts to appropriate it. Human actors as well as material are equally capable of acting because they have agency, and it is this facet that allows the network to learn and evolve.

Collective control: Space around actors is designed with a clear purpose and VCS is no different. We understand better Torvalds reluctance to accept CVS because instead of giving him greater control it actually managed to pull far too much control away from the leader. BK with its pull and push strategy gave the impression of greater collective control yet it provided the facility to Torvalds to watch over the entire collective.

Collective ownership: A VCS is useful to help restore the balance of ownership rights within the collective. Legally [via the GPL] all actors have ownership to some part of the code yet Torvalds emerges as the owner. The use of a VCS redresses this imbalance and supports the creation and continuation of the collective through a feeling of ownership and control.

[239] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.2/0121.html

Chapter 8
Learning and Organizing in the Linux Collectif

Introduction

In the previous chapter three themes of time/space, sustaining the assemblage and machine agency were discussed. In this chapter they are brought together through the overarching theme of the transparency of learning and organizing. VCS, and its ability to manipulate time and space is a necessity for the collective and makes collective learning and organizing so much more fluid. The collective is held together in part by a machination, VCS. The collective's ability to learn holds it together expressed principally through the code. Together, the VCS and code are able to manipulate the collective, as actors they have agency ("the capacity to make a difference" (Rose and Jones 2004).

These concepts are key to the understanding of the final theme discussed in this chapter. The memos give rise to three main ideas that are intertwined and interrelated. The first is transparency – this section discusses how the license gives transparency to learning and organizing, and just as it lets the researcher in so too it lets the members of the collective in. The second concept is organizing, and this section explains how organizing is a technical domain in OS, but our understanding of the technical is balanced by the social structures and governance. We then explain learning and organizing in OS as inseparable concepts because they are inseparable phenomenon. It may appear that learning and organizing are treated apart. It is pragmatic to structure the chapter in this way but we are clear that learning and organizing are intertwined concepts – a duality, a performative duality. Third, we have the learning memo exploring processual change in all actors, human and non-human – which expresses collective learning.

This chapter proposes an argument which begins with a transparent license (thus a transparent infrastructure, governance model and code) upon which a collective (comprised of human, non-human and hybrid actors) is sustained through the intertwined process of learning and organizing that holds the collective together and allows it to face the futures.

Transparent Learning and Organizing

Transparency

This section focuses on the collective's understanding of transparency and how this affects its approach to learning and organizing. We begin by defining transparency and continue with the four sub-sections that together show the significance of transparency in open source collectives. As we have argued before, open source is premised on a particular kind of license and it is the license that in a large way dictates the level of openness (transparency) permitted for the chosen project. The need for transparency for learning and organizing is explained through various examples including metadata, and the need for a historical log. Finally the relevance of material transparency is highlighted to indicate that collective sense-making can only be considered "collective" when material (non-human) actors are given appropriate attention and significance.

The argument in the transparency section is that version control software allows both software and software creators to become obvious which creates an environment of openness. However, this openness is predicated to a great extent on the licence of the software but also the process, in this case version control software.

License Transparency

The license (Constitution) is the basis of transparency in a software process and in this section we get a glimpse of the practical implications of the chosen Constitution.

Trust through transparency – Some BSD developers believe that the GPL is not needed to create accessible software, and rather that it is about employing some VCS to manage patch coordination, "*this mess makes me happy that I'm running and tracking FreeBSD, where there is a single CVS source for everything (kernel *and* userland) without such issues; not to say that there are never conflicts, but at least it is clear what the single source for the system is, and everyone can view it (see every single source line being changed) and about 80 people can directly commit changes. It creates stability, unity and a steady progress*" (Mutsaers, 1998 – Oct 2nd)[240]. The aim is to have little or no ambiguity and transparency through the use of a VCS in the belief that in the long run the project will be stable and grow. Linux developers believe otherwise as we can see in this reply to Mutsaers by Ts'o, "*BSD folks say that they don't like the GPL because they want to let anybody use their code; yet when the OpenBSD crowd start taking NetBSD patches and applying them to their tree, suddenly they start singing another tune. There was even the time when a developer put a copyright notice on their code saying anybody (except*

[240] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9810.0/0152.html

for this particular other BSD variant) was allowed to use that particular patch file" (Ts'o, 1998 – Oct 2nd)[241]. The BSD collective has fractured many times into a number of smaller and less powerful collectives or networks. Such breakdowns have been caused by political tussles, a lack of trust, and no clear organizing structure which can help lay down rules. There is not more but actually less transparency in BSD. The GPL provides transparency because it forces the code to stay open and accessible to all. According to Ts'o the BSD developers 'pretend' to not like the GPL because they don't want to add restrictions to their code yet in practice they don't want anybody using it and cause arguments when it happens because they feel that credit has been taken from them. If every member is allowed commit rights then there is a chance that they will all want to exercise that right. The BSD collective somewhat proves the latter argument and one needs to pause to assess how democracy affects or can affect the organizing of an open source project. Transparency is essential in order to keep political issues low and trust amongst the actors high. This exchange predates the introduction of BK, the closed source product so an intriguing question to ask is that is it enough to use some VCS, even if it is not an open source VCS, or should all software used in the OS process have a similar licence as the software under creation? BK use was withdrawn from the collective not because the collective had mustered enough OS-enthused energy to create an OS VCS, but because the philosophy of openness in the collective began to endanger BK's closed sourceness.

Transparency of code 'in action' – Transparency comes from being able to see the code and make changes to it and with closed source software that is not easily possible. Transparency is also about being able to use and understand software, "*I don't give a damn if you make a dime off your product. I work on open source projects for my own amusement. If you happen to find a way to make money doing it, more power to you. But don't expect me to care about your pseudo-open-source product's bottom line. Especially because I haven't seen compelling evidence posted to this* proprietary licences. He is looking for transparency of code 'in action' of the artefact, not in another actor claiming that his work is effective.

Transparency breeds diversity – Openness of code has some distinct advantages such as generating diversity of ideas and code by making it possible to follow different trajectories of the code or even a fork,

> "*2.4.5 is 26 meg now. It's time to consider forking the kernel. Alan has already stuck his tippy-toe is that pool, and his toe is fine. The "thou shalt not fork" commandment made sense at one point, when free unix was a lost tribe wandering hungry in the desert. When you have a project with several million users that has a scope that simply doesn't scale, it doesn't. Forking should be done responsibly, and with great joy. As in nature, software success breeds diversity. Linux is far more interesting to me for it's general usefulness and openness, which are inextricably related, than for it's unixness, although unix is certainly beautiful. Alan was going to file for divorce over dev_t. Isn't is funny how estranged couples so often are so much alike?... preserve unixness, openness, ...an open source OS for end-users*

ough. Linux has lost a lot of simplicity since I got into
at is a loss" (Hohensee, 2001 – Jun 24th)[243].

ness and importantly usefulness are linked as Linux strengths. Ideas
which stem from the basic premise of the GPL, are used here to
ze that a breakdown (fork) in the collective would actually be a step
ntation of the collective is usually not desired because it causes a
e base and learning capacity of the collective. However, now that
s many thousands of actors strong and its very size is causing a
rganized break may be just what this collective needs in order to
also mean exploring other possible trajectories that this breakdown
pecially possibilities to unlearn old ways and learn new ones).
can grow as a standard in order to become a better organized
fork (a form of unlearning) will make Linux fracture into a number
which can only strengthen the use and adoption of Linux as a
does build on the OS principles that forks are not encouraged but
problems become insurmountable.

– There are levels of transparency and levels create a group of elite

through and thr
it in '96, and th

Simplicity, transparency, oper
of transparency and openness
provoke the collective to reali
in the right direction. Fragme
degeneration of the knowledg
the Linux kernel collective i
breakdown perhaps a more o
make progress. Progress can
suddenly makes apparent (es
Hohensee claims that Linux
collective. The belief is that a
of enterprising opportunities
standard in the long-run. This
are sometimes welcome when

Collective within a collective

create such a collective within the larger Linux collective, like BitKeeper. The excluded actors are not privy to developer comments and often they have to wait longer for updates to Linux, "*those who now chose to carry out their development using the patch+email method, and prefer to submit everything for discussion on lkml before it gets included are now largely out of the loop. Things just seem to *appear* in the tree now, without much fanfare. That's my impression. Rather than Linux development becoming more open, as I'd hoped with the advent of Bitkeeper, it seems to be turning more in the direction of becoming a closed club. This may be fun if you're a member of the club. Ah well, I'm a 'sorta' club member, why should I complain? All the same, I feel that something we all seemed to be headed towards with unity of purpose is somehow becoming more elusive. Being attacked personally for having this feeling does not help*" (Phillips, 2002 – Apr 19th)[244]. Transparency lent by the GPL needs to be complemented by a transparency of organizing. BK and its 'closed source principles' of inclusion of BK-users but exclusion of all others has created a collective within the larger Linux collective. This detracts from the transparency and

[redacted]

given cvsup, annoncvs etc. The most important point (for me) is that currently no history and no log messages are visible. CVS would solve this, even if only one person commits" (Cracauer, 1998 – Oct 5th)[245]. Transparency and access to code and logs is considered to be the most important element of VCS use and how Linux is organized in the way of commit rights is secondary to the collective. The learning repository idea is again coupled with the need for transparency.

Transparency afforded by metadata – Organizing code through a VCS is encouraged because it provides easier and more direct access to code for the collective which is needed in a distributed development process but it also allows anyone who does access the code more information and a better understanding of why any change was made,

> "*You would have snapshots of a tree, without any of the commentary that usually goes with tree. If someone changes*
> *outp(0x56, 0x60); to*
> *outp(0x76, 0x60);*
> *all you know is that the change has happened. With a good source code control system, you would know why. That is what makes the CVS trees from FreeBSD, at least, so valuable. People tend to document why they made the changes, so you'd know that the MUMBLEFOO bit was added to the initialization sequence to help BRAIN-DEAD-MONKEY keyboards operate properly....*" (Losh, 1996 – Feb 19th)[246].

This making sense of code is very important in collective learning. VCS thus creates a transparent environment of organizing that fosters learning, growth, trust and a learning collective.

Marketplace transparency makes imitation and inspiration possible – Actors reiterate the point that VCS makes possible a marketplace for learning ideas and that code is accessed, among other reasons, for learning purposes "*the whole reason d'arte behind the "open software" process is that it gives people a way to take advantage of the monkey-see-monkey-do effect, a recompense that previously wasn't available at all. Of course I am not expecting my next GNU contribution to pay my bills. Yep, a fine line, but deep. Did they copy your code? Go get some relief... Did they copy your idea? Take the hit and move on... Was "your idea" conceived in a vacuum? If not, bow your head in shame for yelling at others for doing "to you" what you yourself have done in greater orders of magnitude to all those who came before you. (that being building on someone else's ideas, if you haven't kept up...*" (White, 2003 – Apr 8th)[247]. The collective tends to learn by imitation so transparency in the process is paramount. Learning is reflected and manifested in new code that builds on old code, thus old learning begets new learning. The case being built here is that a marketplace builds and organizes open source activity and the understanding of this thesis is

[245] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0548.html
[246] http://www.ussg.iu.edu/hypermail/linux/kernel/9602/0671.html
[247] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0304.1/0257.html

that VCS is the marketplace which organizes and performs the economy of transactions that the actors involved in the open source process explore. Callon draws on his idea of framing to explain how this marketplace functions but adds that we are able, through VCS, to also make sense of the failed as well as the successful transactions.

Transparent Organizing

Transparency of process – Organizing is premised on some level of transparency in the process. If the collective is unable to understand why and how things are done then there will be little trust, which will in turn not promote sharing of code created by actors. Linux was losing patches and this slowed down progress of the entire process. There was no reason given as to why patches were not being added or even studied. This caused a breakdown in the collective. If the OPP is not able to hold the collective together and loses their trust then the network breaks down. If the collective performs its role in the network then there are expectations from the OPP [Torvalds] to do likewise. "*As a 'black helicopter devotee' I'll state I have no desire to see anything resembling a 'core team' or the bullshit politics that most certainly will come with it. Things just need to become ORGANIZED. There needs to be a solid system for supplying patches and publishing kernels and Linus needs to USE IT. I've supplied a few small patches and a new feature. I have NEVER heard anything back on them. The closest I got was when I submitted the 2.0 version to Alan Cox...and ended with Alan waiting to hear back from Linus. : P Until Linus gets his shit together I'm apprehensive about working on the kernel anymore except for things I can ship directly back to Alan. (Which generally does not mean new features; what I'd like to play with.)*" (Cinege, 1998 – Oct 1st)[248]. To be organized for this developer means to have a one-actor leader and not a core team. The core team just breaks out into political rivalry and fights that cause more confusion. Some help is needed but from technology and perhaps not another human actor. The misguided belief of this developer is that technology does not bring any politics with it. Everything has politics (Latour 2005a), and the politics of technology are far more insidious because they are less apparent, and not expected. Technology is created by humans and hybrids and the creators embed their own politics into the code and design of technology. With OS software these politics are a bit more apparent because the code is visible and accessible. It is also read by experienced developers and others who want to learn and are thus more aware of what the code is capable of doing.

Organizing patch submission (collecting) – Transparency, organizing and learning are interdependent. It is a combination of all three that keeps the collective assembled together,

[248] http://www.ussg.iu.edu/hypermail/linux/kernel/9810.0/0175.html

fosters and maintains learning, allows for innovation and maintains trust. We see a breakdown of the above through an exchange between Torvalds and the collective, in particular here with Ts'o,

> "*The reason I'm disappointed is that vger in particular has been acting as a "buffer" between me and bug-fixes, so that now we're in the situation that there are obviously bugs, and there are obviously bug-fixes, but I don't see it as such, I only see this humongous patch. I don't know what it fixes, because vger has kept me out of the loop, and quite frankly I don't have the time to look at several hundred kilobytes of compressed patches by hand. And I refuse to apply patches that I don't feel comfortable with (Torvalds, 1998 – Sept 29th)*[249].
>
> *Linus, To be fair to the vger people, one of the problems which the vger CVS tree is trying to fix is that sometimes you don't take patches very quickly. There's been at least one set of tty patches which I had to send you two or three times before you finally accepted it --- and they were short patches (1-3 line changes in 4 files), and with a full explanation of what it did. Heck, in the most recent case you and Alan and I discussed different approaches for solving this problem before I even started coding the patch. Yet I had to resubmit the patch and the explanation at least *THREE* times, over the course of 2-3 weeks, before I finally got a response out of you. And this was for a utterly uncontroversial patch! And this was not the first time I've had to resend patches 2 or 3 times, either. Is Martin going to have to send you each driver update 2 or 3 times before they finally get accepted?!? Now, I've had to do this enough times that I just simply consider it par for the course, and the changes are important enough that I'm willing to repeatedly resend you patches again and again. But the risk is that other people who are not so persistent may give up, and we therefore end up seeing patches get dropped. So we have a problem, and perhaps vger isn't the best solution. Maybe the linux-patches web page is a better solution. But before you go slamming the vger folks because this patch batching effect which you don't like, it might be nice for you to acknowledge that some of your bandwidth constraints may have contributed to the problem, and they were simply trying to find a way around it. If the vger CVS tree doesn't work, fine, we can try to find another solution. But before we can find an effective solution, it often helps to admit that we have a problem*" (Ts'o, 1998 – Sept 29th)[250].

Transparency of roles – Part of the problem with Linux kernel development is that not many roles for any actor are clear-cut. As this developer points out the role of Torvalds too is ambiguous. Torvalds seems reluctant to give up some of the key roles like integration maintainer due to a lack of trust in the collective's ability to carry out that function faithfully. He is able to delegate work but needs to build very high levels of trust before he allows others into his closer network,

> "*Torvalds - One "patch penguin" scales no better than I do. In fact, I will claim that most of them scale a whole lot worse.*
> *Landley - But Alan doesn't, and Dave Jones (with enough experience) shouldn't. You have architecture duties. You're worried about the future of the code. You have to understand just about everybody's subsystem. An integration maintainer would NOT be making any major architectural decisions, they would be integrating the code from the maintainers, collecting the patches for the unmaintained areas of code, and resolving issues between maintainers that are purely implementation details.*
> *Torvalds - The fact is, we've had "patch penguins" pretty much forever, and they are called subsystem maintainers. They maintain their own subsystem, ie people like David Miller (networking), Kai Germaschewski (ISDN), Greg KH (USB), Ben Collins (firewire), Al Viro (VFS), Andrew Morton (ext3), Ingo Molnar (scheduler), Jeff Garzik (network drivers) etc etc.*
> *Landley - Query: Do you not believe you have been dropping a significant number of good patches on the floor?*
> *Torvalds - Good maintainers are hard to find. Getting more of them helps, but at some point it can actually be more useful to help the existing_ ones. I've got about ten-twenty people I really trust, and quite frankly, the way people work is hardcoded in our DNA. Nobody "really trusts" hundreds of people. The way to make these things scale out more is to increase the network of trust not by trying to push it on me, but by making it more of a _network_, not a star-topology around me.*
> *Landley - I'm talking about clarifying the current ad-hoc development process. Formalizing an existing de facto position so people farther out in the development process know what to do and where to go*" (Landley and Torvalds, 2002 – Jan 28th)[251].

Torvalds explains that he wants a network of actors that are related and interacting on the basis of close trust. Connecting actors to other right actors is the solution to the Linux backlog problem, or so Torvalds believes as he is not at this point willing to consider any material actor intruding into his development process. He doesn't want any more maintainers as he says that he is not able to trust everybody but he is happy with his maintainers organizing their trust with other developers and then pouring that learning back to Torvalds for affirmation. This conversation between Torvalds and Landley is indicative of the organizing problems the Linux development process faced. There was no transparency in the process because the feedback loop was broken and the collective did not feel like a collective anymore because their OPP, Torvalds, had suddenly fallen

[251] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0201.3/1074.html

out of contact. Torvalds holds the network together and when he stops responding the collective begins to fragment and splinter. Torvalds insists it is not about trusting more people but about trying to create networks within networks that will bind the collective together stronger. Landley feels that this may well be true but actors need to have specific roles in order to organize more effectively and trust more openly. If roles are clear-cut then so is the line of responsibility but it may well be that Torvalds prefers a more opaque (flexible) structure.

Politics and transparency – Transparency and openness of Linux is linked with its lack of strict and clear-cut ways of organizing and is highlighted as the real strength and reason for survival of the Linux kernel project. The belief is that there are already too many versions of Linux flying around and more are not needed. There is a need to consolidate the collective rather than let it disintegrate into numerous smaller and less strong collectives. And helping Torvalds to do that is the way forward rather than letting his control over the kernel disintegrate by adopting a committee structure. Too much organizing takes away from the in-built slack of the Linux kernel

[redacted]

Linux kernel has shared ownership, in the GPL sense, so this spokesman is remonstrating against any small group taking control. There is little faith in a purely human remedy and so indirectly, some form of material intervention or inclusion is endorsed. Speaking against formal rules and bureaucracy, it seems that the collective prefers more fluid organizing which is emergent, open and accessible in process but ambiguous and nebulous at the same time. This may promote collective control of the Linux kernel and a more democratic approach to organizing. Landley, above, asks for greater organizing and clearer roles but Schmeidehausen believes that the strength of Linux lies in this lack of organizing (or a new form of organizing).

Material Transparency (usability of artefact)

Transparent control over code – VCS makes possible traceability of patches but certain VCS, like BK, are able to fold the chronological collection of patches to disconnect them from any one moment in time and that is why there is "*no excuse to keep the kernel source without a version control system where people can see the history and log messages for changes. And the amount of*

[redacted]

transparency would again encourage trust in the organizing of Linux. The collective needs access to such material so that they can all read a patch they need without depending on anything other than the VCS and independent of time. Transparency and openness for all leads to a stronger and

[redacted]

But Alan had a bad week, some butthead managed to slip something into the alan branch, so his branch looks like good.1 - good.2 - bad.1 - good.3 Linus, being the smart guy that he is, reads the unified diffs which he insisted be at the front of each BK patch, and notices that there is a bad patch in there. So he whips up the gui tool, finds that part of the patch, and says "OK, I'll take the three good ones but exclude the bad one". And the system does. The really cool part is that BK has a sort of #include facility which works across branches - the data for each patch enters the revision control system exactly once." (McVoy, 1999 – Sept 27th)[254]. VCS creates the sort of openness to the code that can be understood, at least by this actor, to generate higher levels of security for the Linux kernel code. There is also greater transparency for BK-users but far less for others.

Summary

The main claim of transparency has been explored and in conclusion we add that collective sense-making and organizing requires a level of transparency in the infrastructure, governance, and actors that allows openness such that learning and organizing can occur. Collective sense-making is explained by Brozefsky (1997) when he sums up the basic need for a VCS, "*CVS could be useful to the Linux kernel project as a whole possibly. I'm not sure how this would fit into the usual Linux methodologies for adding things to the kernel, but it would make things easier for developers to watch code changes as they happen, see where collisions with their own patches occur as the kernel progresses, and a few other things*" (Brozefsky, 1997 – Apr 10th)[255]. The becoming of different trajectories becomes less 'opaque' when a VCS is included into the collective. This message reflects the synergy of the three ideas of transparency, organizing and learning. The Linux kernel is a strong collective because it is a learning collective. It is the ability to see and read the code, and understand what happens to it and why, that helps the collective sense-make. This developer importantly points out that actors are able to see where collisions or breakdowns happen and learn to counter such challenges. Survival of the kernel reveals an ability to learn and adapt in the Linux kernel collective. Bateson would argue (as would Weick (1979), Patriotta (2003, 2004) and Latour (1991, 1994, 1995)) that a breakdown stimulates reflection which in turn often creates an unpacking of the issue and thus some level of learning.

Organizing

The organizing memo leads to a narrative that focuses more on the technical side. The human side of organizing has been explored (Crowston and Howison 2004, Moon and Sproull 2000) as have some aspects of technical organizing (Hemetsberger and Reinhardt 2004) but the non-human as a

254 http://www.uwsg.indiana.edu/hypermail/linux/kernel/9909.3/0547.html
255 http://www.ussg.iu.edu/hypermail/linux/kernel/9704.1/0125.html

social phenomenon in the OS literature has not been studied, thus the social aspects of technical organizing (collective organizing) is explored here. The technicalities of VCS quickly lead to a framing of the technical and social governance structures side-by-side to indicate how each mirrors the other. The empirical data draws our attention strongly back to the link between learning and organizing as we find it impossible to speak of one without exploring the other. The last section of this memo is thus devoted to the intertwined nature of both where we draw out the link from an organizing perspective. In the last memo, learning, we return to this point but this time from a learning perspective.

Weick's concept of organizing emphasizes that there is little stability in any process. It is all about changing and organizing not organization or organized. Actors strive for stability but it is very fleeting. The Linux collective is also in a constant state of flux,

> "*The problem is that some complicated changes were made. Those affect the whole system. So while 2.3.8 may work for a whole bunch of people we can be pretty sure that it will break for others. Sure, Linus is optimistic about it. If I make a graphical representation of the trajectory from 2.3.0 to 2.4.0, it could look like this: Read the "X" as "change made" and the "---" as "change still has some problems".*
> *2.3.0 .8 2.3.99 2.4.0*
> *a X---------*
> *b X-----*
> *c X---------------------------*
> *d X------------------------------*
> *e X------*
> *f X-------------*
> *So, as you can see, there is no moment in the development where everything is sorted out*" (Wolff, 1999 – Jun 23rd)[256].

Software development is a process and Wolff (1999) makes it clear that nothing is ever 'sorted out' or organized. He speaks of different trajectories of the code and how any patch sent to [illegible]

[illegible] ns. Each step is a possible trajectory and if it gains critical mass and the inscription grows then that trajectory will become the stronger translation. Each change has some positive effect but often changes bring some unintended consequences. Attempts at solving the new issues then force the process of organizing and learning to continue.

Technical Organizing

Mechanizing organizing – The significant breakdown faced by the Linux collective [when Torvalds began to drop patches and ignore his developers] forced the collective to understand that the current form of organizing is not productive. They began brainstorming for alternative approaches to organizing Linux development. The discourse of searching amongst alternatives was a collective approach to both learning and organizing. A strong need for the collective to become more of a hybrid was voiced,

> "*Maybe we have to come to terms with the fact that it is possible for a projects size to just require that there is some mechanism to "mechanize" the development process. This can come in two forms:*
> *1) Something like Jitterbug*
> *2) Letting more than 1 person be the only ones who can make direct changes to the source*
> *#2 is not an option to Linus, we did #1 and it worked, now #1 is gone and we have problems again*" (Miller, 1998 – Sept 30th)[257].

Basically, this is demand for a VCS because it will allow for both organizing and learning. The collective has understood that it needs to resolve the current crisis in organizing. This can be understood as level 2 learning (Bateson 1972) which is the ability to adapt behaviour when the context is changed and is a result of reflection and paradox resolution. It also shows that the new

[redacted]

alpha again, thereby killing and restarting the branch. Oh. And that's 3 branches and not half a dozen. ;) Hope that made sense. :)" (CaT, 1999 – Jun 24th)[258]. VCS is capable of managing numerous branches of code running in parallel and each branch can be folded to create a singular coherent one in an organized way. This is relatively straightforward for a VCS but a human actor is more vulnerable to error. Software makes sure that nothing valuable is overwritten and multiple trajectories are possible to exist concurrently. Each trajectory or branch is a possible learning pathway and hypothetically, the more branches there are the greater the ability to innovate for the collective.

Feedback as a crucial element of organizing – Communication is essential in organizing [Weick's idea of the double interact]. If Torvalds had sent some form of feedback to the collective then the breakdown in the collective would not have escalated as fast, "*all these stories about repeated patch resubmissions being necessary sound all too plausible in the absence of any official, publicly visible patch queue with your annotations on it. I think it would help matters all around if there were a web page somewhere listing pending patches and even as little as a one-line*

[redacted]

comment from you as to where they are in the integration process. It might even help you organize your own work better" (Raymond, 1998 – Sept 30th)[259]. Actor network theory also speaks of interaction and a network of relationships. Transparent or visible feedback and interaction is a crucial element of organizing. The collective needs transparency as it helps to redress the imbalance in control between itself and the leader. A collective where access to information like patches, versions, comments and feedback is accessible to everyone allows all actors to learn and express their ideas. A breakdown in the current system has awakened the collective to the lack of interaction and thus breakdowns should be seen as a way to solve a problem rather than as the problem itself. It is symptomatic of some other issue which actors have been oblivious too but it forces open the blackbox to reveal the problem.

Technology Mirrors Collective Organizing

The Linux kernel development process is about the code. Code is an important macro actor and we often hear it 'speak' through various spokesman and via its functionality. Code needs to survive and survival depends on constant change and evolution. When developers are outraged that patches are being dropped we have strong indication that code has enrolled these actors into believing that their survival depends on the long term existence of the code. In such instances Torvalds is not looked upon too kindly.

Refining organizing – Code is organized to allow some measure of stability but enough slack is allowed so as not to stifle creativity and innovation. The period when Torvalds began ignoring patches and developers the collective made attempts to organize the collective to function in the event that Torvalds left. We saw a greater need for organizing (less slack). "*That is good for identifying which parts of the linux kernel are beta (or alpha quality). I am talking about a stamp of approval from the kernel developers for the overall kernel. Right now there are only 2 labels -- Development and Stable. I am taking about refining it a bit more-- development, beta and stable. At the risk of sounding bureaucratic, let me suggest a scheme. I think we should wait for x days (a week, 10 days pick your choice) before a beta version is declared stable. It just make the life of users a little easier if the software itself gives more information about its quality*" (Juvvadi, 1999 – Jun 22nd)[260]. If code has more authority to speak then this brings about disintermediation in the collective and makes the middle man, Torvalds, less indispensable. Code speaks through its quality and usability. If it is elegant code then it takes control from other actors around it, be they human, technological or hybrids. Patches have to work together so if code were allowed to organize itself it could be more efficient. The act of organizing is linked to non-human actors in OS. If code is stable and works then the programme will run smoothly. When code works it shows that a certain trajectory can and should be pursued, thus code leads the way to organizing.

Macro actor decides – Most open source projects are a strange mix of democratic plus rather autocratic structures. Linux is no exception, indeed it is a good example of a mixed form of structure and governance. The structure of Linux is informed by the governance style of the project and vice versa. Linux is considered to have a dictatorship style of governance by its collective and there is open approval of this style of rule, "*the essential problem with a core team is specifying how you decide who is a member of the core team, and who is not (which is an extremely political act), and how do you resolve disputes within the core team. In the Linux model, one of the things which works is that we don't have the argument of who is on the core team, because there is no such thing (in all of the *BSD cases, the splits occurred when someone was thrown out of the core team). Also, we have a relative simply dispute resolution mechanism --- Linus decides. Richard and Larry and others may argue about scheduling changes, or Richard and I may argue about the appropriateness of devfs, but ultimately Linus gets to make the final decision. This all boils down to the old saying that the benevolent dictator is the best form of government --- there's only one problem: finding the benevolent dictator. Linus has, up till now, served as a very good benevolent dictator. It may be that the job has been putting much pressure on him, and we need to find ways of relieving this pressure, or otherwise solving the problem. I also see the NetBSD and OpenBSD, and the conflicts which produced them, and that's also part of*

[260] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9906.2/1440.html

*the *BSD model. Note that for better or for worse, the organization of the various *BSD's aren't all that different. The personalities involved seem to be what makes the most amount of difference*" (Ts'o, 1998 – Oct 2nd)[261]. The single, strong OPP style of governing is clearly chosen over and above the more 'democratic' BSD style. A single OPP has a better chance of becoming and sustaining an assemblage with a critical mass of followers, and that is why forking of projects is not encouraged in open source. GPL promotes a democratic culture, but this raises the question of why Linux is not a truer democracy. Is it all about making organizing easier, and is organizing made easy when one person makes all the serious decisions rather than a committee of experts? Linux shows us both sides of the argument. It is very successful in so many organizing methods yet the premise of a dictatorship puts far too much pressure on a single actor. Torvalds needs help, but help that suits him and doesn't make him feel powerless and nor gives too much power to the collective. There is thus a need for a particular and tailored VCS – BK.

Distributed controlled hierarchy of VCS and Linux – VCS reflects the hierarchy of the governance structure of the collective. The need for a VCS stems from a need for organizing, but the interesting point is what sort of organizing does any particular VCS allow. BK allows the collective to do a 'BK pull' and thus clone the code and work with it freely, and in that respect it is very distributed in structure, yet when it comes to a 'BK push' of any changes back to BK and Torvalds then the situation is more controlled,

> "*A big part of using BitKeeper is organizing the various trees you have on your local disk, and organizing the flow of changes among those trees, and remote trees. Since a "bk push" sends all changes not in the target tree, and since a "bk pull" receives all changes not in the source tree, you want to make sure you are only pushing specific changes to the desired tree, not all changes from "peer parent" trees. One would typically work on only one "theme" at a time, either vm-hacks or bugfixes or filesys, keeping those changes isolated in their own tree during development, and only merge the isolated with other changes when going upstream (to Linus or other maintainers) or downstream (to your "union" trees, like testing-and-validation above). It should be noted that some of this separation is not just recommended practice, it's actually [for now] -enforced- by BitKeeper. BitKeeper requires that changesets maintain a certain order, which is the reason that "bk push" sends all local changesets the remote doesn't have. This separation may look like a lot of wasted disk space at first, but it helps when two unrelated changes may "pollute" the same area of code, or don't follow the same pace of development, or any other of the standard reasons why one creates a development branch.*
> *Small development branches (clones) will appear and disappear..*
> *While long-term branches will parallel a tree (or trees), with period merge points. In this first example, we pull from a tree (pulls, "\")*

[261] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9810.0/0257.html

periodically, such occurs when tracking changes in a vendor tree, never pushing changes back up the line...
And then a more common case in Linux kernel development, a long term branch with periodic merges back into the tree (pushes, "/"):
Submitting Changes to Linus - There's a bit of an art, or style, of submitting changes to Linus. Since Linus's tree is now fully integrated into the distributed BitKeeper system, there are several prerequisites to properly submitting a BitKeeper change." (Garzik, 2002 – Feb 21st)[262].

This actor talks about working towards being upstream or downstream depending on whether you are respectively trying to approach Torvalds with your patch or working back towards your own base code branch. Each branch has the sort of slack required to let each actor fiddle with the code but the more upstream you go the less slack there is as you are trying to fit into some other actor's branch so must work within that area and the trajectories found suitable by that actor. The more distributed the VCS the more slack there is in the network lower down in the branches but the moment a smaller network clashes with a larger, stronger one, then the smaller one must adapt.

ıt match the infrastructure (Conway's
ment of organizing or even a product of
ities of various trajectories that can be
; at organizing and suggestions for

ɔantheon of the gods by
g whatsoever taken out it
ose who complain in the

tor to clearly designate
kernel.
ɔund by his approval or
t kernel" (Talbot-Wilson,

n action, exploring a potential pathway.
ıcy in the development process and to
reveal the collective's need for some

.060.html
elvin Conway, who introduced the idea in 1968.
ained to produce designs which are copies of the
y piece of software reflects the organizational

Becoming trajectories – The governance of Linux mus
Law)[263], but also the way the network is unfolding. An ele
a particular sort of organizing is that it raises the possibil
pursued. The Linux collective made several attempts
improvement were many,

"*The way forward may be threefold.*
(a) Promote Linus even higher into the p
agreeing here that when he requires anythin
is done without any discussion, and that th
list about it are to be ignored.
(b) Expand the role of the Changes edi
unapproved vger functionality in the official
(c) Agree that in any case Linus is not b
acceptance of any patch to the developmen
1998 – Sept 30th)[264].

This is one of the many attempts at organizing, *becoming i*
The collective makes yet another attempt to cut inefficie
become less reliant on Torvalds. Such organizing efforts

[262] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0202.2/1
[263] **Conway's Law** is an adage named after computer programmer M
Conway's Law states that organizations which design systems are constr
communication structures of these organizations. In other words, an
structure that produced it (Wikipedia).

stability – but in order to secure stability we need to keep changing (Weick and Westley's (1996) oxymoron). The more structured the collective becomes, the less slack is available but actors are desperate for some stability and are willing to a degree to give up the freedom to improvise.

Desire for stability – The desire and need for stability is more forcefully expressed by Brian (1999) when he complains that "*couldn't the "incomplete or experimental" sections of 2.2.x kernels already be considered Beta? I mean, I think the beta channel already exists in the current versioning system*" (Brian, 1999 – Jun 22nd)[265]. In the process of organizing the Linux model has created greater possibility for both stability and learning. The stable branch of any version release is the outcome of trying to achieve stability or organization. The experimental branch makes every room for slack to be built into the system so that learning and innovation can be allowed to happen. When this learning is consolidated enough [well inscribed] then it is added to the stable branch. The Linux versioning system is a product of a greater level of organizing that allows more micro organizing and learning to flourish at the same time without impeding either stability or learning. VCS facilitates such organizing and indeed in the case of Linux, makes it possible.

Evolving organizing – Actor Network Theory and Bateson's levels of learning concepts stress the evolutionary aspect of reality creation. The Linux collective is also quite clear on the evolving nature of code organizing, "*The current linux versioning scheme is very clever. It underscores the point that a software project never really completes. It only evolves. This strategy has served us very well in the past, but I do think there is a need for a change. At present, the code evolves in two branches -- development (odd number) and stable (even number). I think we should at least split it into three branches -- development, beta, and stable.*
Development --- kernel developers only
Beta --- Adventorous users only
Stable -- No serious bug reported in the past x days
Of course, we get into the problem of not getting enough users to pound on beta versions. But I think the number of linux users is large enough that bug reporting will not be a problem" (Juvvadi, 1999 – Jun 22nd)[266].

VCS is designed in a way that it provides ample room for evolving organizing. It is complemented by the release strategy of Linux where there is both an experimental and a stable branch of development. This model of versioning is useful not only for utilizing the software produced but also a way of organizing the developers in the process of development. The important point Juvvadi makes is that the Linux versioning system is also an evolving one and

[265] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9906.2/1417.html
[266] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9906.2/1368.html

there is no clear stability or end. VCS makes possible parallel dimensions of code. Nothing is ever stable, the only constant for software, as for the collective at large, is change. A form of organizing where there are more rules (organizing) could eventually detract from learning, however, too many trajectories of ideas could dilute the critical mass of actors related to any OPP. The situation could arise that there are numerous parallel translations and none of them strong enough to hold the collective together. Development (learning) is not more important than organizing because it is the process of organizing and learning together, in a reflexive way that allows for both. This message thus brings the idea of learning and organizing together very well. We can understand, as Weick points out, that learning and organizing often go hand in hand and if either gets out of control then there is a strong possibility of a breakdown. The aim is to keep a balance between organizing and learning for any 'healthy' collective to continue and thrive at a stable pace. This is not to say that major breakdowns cannot be turned around into yet more learning/organizing.

Organizing Meets Learning

'Bad' mutable mobiles – We find that learning and organizing are linked at a more micro level as well. This example illustrates our claim. Developers are unhappy about the lack of transparency in the development process "*I think the linux-patches@samba.anu.edu.au idea is much better: That way not just Linus but other people too have a chance to look and try the patches, and bad ones will get marked by comments saying so. This should filter out much useless stuff. What happened to that?*" (von Brand, 1998 – Sept 30th)[267]. An open approach is a requirement because of the transparency it lends the process. Comments and metadata (transparency in the process) provide information on why certain patches were accepted while others rejected. A website may not be as flexible as a VCS but it will provide the needed transparency. Feedback in this model is what ANT would describe as the circulating reference and the evolving centre of calculation. The circulating reference in this case is the idea which is passed on, but as it changes 'hands' it lets go of certain properties and takes on new ones from the actors it comes into contact. This way it grows and adapts to the needs of the collective. It is a centre of calculation that is being added to or shedding attributes. This is not an immutable mobile but rather a mutable one because it changes as it moves (Mol and Law 1994, Moser and Law 2006). Transparency makes apparent 'bad' mutables so that they can be forced to evolve in an appropriate way or discarded entirely. How such fluid mutables flow and to which actor is clearly linked to organizing and make *becoming* apparent through the trail they leave. A mutable is learning trying to manifest itself and as it flows through the network it evolves in a way that is allowed by both the structure and actors.

[267] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0987.html

Organizing and learning – When a certain level of learning is achieved then the collective direct the code development towards stability, no matter how short-lived, "*it is unstable development kernel which is becoming stable since it is nearing end of development cycle for 2.3.x series*" (Sulmicki, 2000 – May 10th)[268]. Stability or irreversibility is needed in order to hold the collective together, and in this case the collective includes the users of Linux [business users and novice users]. A mutable mobile is *forced* into an immutable stage to stop short too many off-shooting trajectories. 'End' in the sentence implies death or no more change. Learning and organizing are continuous but actors attempt to set short goals and thus create artificial stability points in the development process. This provides space for reflection and some manner of planning for the future.

Ordering patches for sense-making – VCS orders patches and organizes them in a way to help the collective make sense of the code created,

> "*The whole point of having things under revision control is so that you can browse them, in the order that the events happened, not in some random order which corresponds to when you happened to find some old kernel. As to the tree itself, I'll take responsibility for checking in what we have archived to date and providing that tree as a basis for future work. It's true that if you are at version 1.100 in your copy of the source and patch which consists of the diffs from 1.200 .. 1.203 comes by, BitKeeper will not allow you to patch that into your tree. Under no circumstances, not even if Cindy Crawford showed up and begged, would I change that behaviour (she's welcome to show up and try to convince me, however :-) The reason is that "the source" is a distributed thing. There are copies of the same files all over the world. You guys all want that managed well so that you can do your work in a distributed fashion with a minimal amount of fuss. When Linus edits diffs, what is in the tree is your diffs plus Linus' changes, *as one delta*. And that delta is different than the one delta you made in your tree. The correct set of changes is whatever you did plus whatever Linus did as two distinct events. Linus gets to work the way he wants and we can still preserve the invariants required to have a distributed source base*" (McVoy, 1998 – Nov 24th)[269].

Patches and changes are not sent in any order but for each branch of the Linux code the chosen VCS has to order and sort the work in some fashion that reflects evolutionary learning. If one patch builds upon another then these patches must be ordered that way too. This does not detract it accessible, in other words it must be organized. The ability to track any change back to its root is the sort of traceability required to build a collective knowledge base or repository. VCS is thus capable of organizing from random to some meaningful order. It is intelligent software and capable of evolving and learning.

Excessive slack and little organizing – Some amount of slack in every collective is needed if it wants to retain the flexibility to evolve and learn yet too much slack and a lack of rules that accompany such a state often lead to a breakdown. This is clearly evident in the Linux kernel collective as can be seen in the discourse below between Ts'o, Torvalds and Wolff (1998),

> "Theodore Ts'o: *Yet I had to resubmit the patch and the explanation at least *THREE* times, over the course of 2-3 weeks, before I finally got a response out of you. And this was for a utterly uncontroversial patch!*
>
> Linus Torvalds: *Note that if some person cannot be bothered to re-submit, I don't WANT the patch. Anybody who is not willing to take that much care of his patches that he can't maintain it while I haven't accepted it, I don't want to accept patches from anyway.*
>
> Rogier Wolff: *Linus, the problem is that when I submit a patch, I don't want to unnecessarily overload you by resubmitting on too short a notice. So I wait, and wait. Nothing happens. The way things work right now (almost never any feedback) you'll get the same bad patches to look at over and over again, especially now you've begged maintainers to be persistent in sending you patches. I'm willing to take as much workload off you as possible, but you do have to be clear in how you want us to do that*" (Wolff, Ts'o and Torvalds, 1998 – Sept 30th)[270].

Without any rules of engagement and relationship building organizing is breaking down. There is real confusion and hesitancy in the collective and so we have Wolff ask for a 'Linus manual' to set out criteria that the collective can follow with some hope of patch acceptance. The Linux FAQ page no longer covers the unexpected situation of Torvalds ignoring both developers and patches and so is a defunct way of organizing. Patches are as unhappy as the developers and speak aloud through spokesmen like Ts'o and Wolff. The current model of organizing has broken down and now there is too much inefficiency in the system to allow any progress of the project. Excessive slack has converted to inefficiency and lead to disorganization. This inevitably won't allow for much coordinated learning. We see the collective *coping to learn* in the face of Torvalds' capricious behaviour. It is however, through this breakdown, and others, that the collective 'learns' to eventually organize better.

[270] http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0871.html

Summary

The main purpose of the organizing section is to tease the intertwined nature of learning and organizing which is more fully explored in the next section titled learning. We show how the collective is able to organize learning, or what we term as *learning through organizing*. Even a brief glimpse at how VCS organizes development and the collective will make clear that VCS not only coordinates and organizes but in this process organizes learning. Most importantly however, VCS makes *learning through organizing* possible,

> "*Check in the new source on the same branch as the lowest and _closest_version number. If said version number is not at the head of the branch in question, branch at said version number.*
> *I hope this made sense. Good luck w/ the Linux history repository*"
> (Alexander, 1998 – Nov 24th)[271].

Each branch is labelled and order is being created. The collective learns as it organizes itself, it is a way of organizing learning or *learning through organizing.*

Learning

Drawing on Bateson's work of learning (1972) (combined with Weick's (1995, 1988, 2001) ideas of sense-making) we have focused on how the Linux collective understands learning. This section explores learning as change. It then goes on to make evident how learning and organizing is a process which is ongoing and never ending. The subject of learning in technology is considered and some similarities in the process of learning and evolving of human and non-human actors is reflected on. The last section under the learning memo emphasizes the move from individual to collective learning and organizing. The key aspect we would like to draw the reader's attention to is how at this point of the analysis we find it very difficult to explore or speak of organizing without learning and vice versa. This strengthens our claim of how intertwined both concepts are in practice.

Learning as Change

Changing – Bateson defines learning as a change in behaviour (Bateson 1972), and ANT discusses performative learning through change in inscriptions and centres of calculation (Latour 1987). The Linux kernel collective describes learning through change as well, "*that's an interesting growth in file size over the last 7.2 years, and really shows how something can*

[271] http://www.uwsg.indiana.edu/hypermail/linux/kernel/9811.3/0301.html

change..." (Williams, 1998 – Nov 22nd)[272]. Technology [the code] is learning as is the network at large, including the developers. The entire collective learns through changes in the code.

Changing is becoming of solving problems – The collective stresses the constructive aspects of change in the behaviour of code despite the fact that not all change is positive, "*fixing the problems will mean changing the current system. If you don't change anything, you wont fix anything*" (Miller, 1999 – Sept 28th)[273]. Two points need discussion here, one is the clear message that a change in code entails a change in the behaviour of the code and software functionality, and two, that there are many potential ways or *becoming* before the correct trajectory becomes apparent. When a problem becomes noticeable one learns that the approach taken is not working and must be changed. The realization of the problem is learning Level 1 (Bateson 1972) and efforts to solve it can lead to Level 2 learning. 'Scratching a personal itch' amounts to Level 1 learning and though a very simple example of a breakdown it is one that developers face very often.

another part. Such erroneous trajectories are windows to learning. Collective learning can be very helpful in such situations because 'given enough eyeballs, all bugs are shallow', "*As someone who maintains a fairly large source base I get nervous when people tell me they need a debugger to work on the code. Why? Because if you really need that it is EXTREMELY likely that you don't understand the code. If you don't understand the code then YOU SHOULDN'T BE CHANGING IT. It is infuriating to have a section of tricky code that used to work, you turn your back, only to find that someone made a "simple change" which seems to work but actually makes things worse and invariably seems to break the code in a far more subtle way*" (McVoy, 2002 – Sept 15th)[275].

Remembering and learning – A vital element of learning is memory. Technology is capable of retention and making decisions based on an ability to distinguish and differentiate, "*after that the SCM remembers that this merge was done already and doesn't ask me to do it again when I move my code base to the next official kernel version*" (Aliagas, 2002 – Mar 8th)[276]. VCS must adapt alongside all the code changes in order to keep abreast and perform effectively. This ability to keep up and evolve as the environment changes is learning in technology. Technology is capable of learning but the more interesting question to raise is what level of learning is technology capable of achieving? Does technology possess the requisite agency to be able to achieve Level 2 of learning and above? [More on this below]

Learning as Process/Performance

Learning by doing is learning by organizing – Learning is performative as is organizing "*the above tells me you haven't tried BK and stayed with it very long. It's been intuitive for doing simple things and almost the same is true for complicated things. CVS for the kernel is a nightmare compared to BK. I've had to go back to CVS for some linux/mips work (linux/mips is managed mostly through CVS, not my choice) and it's been a mess.... If it takes a man-year to learn BK, that person shouldn't be doing kernel work. It's beyond them. Putting shoes on would be a monumental effort of mental power for them. BK isn't CVS, RCS or any other revision control software. It aims to do things differently - and better. There is some learning, but it is far far from a year. It took me 2 hours to get to the point where I was competently creating trees and giving other people access to our trees. That includes merging in Linus' patch with the _weird_ pre-patch format - that's not such an easy thing. My experience isn't unique. All of our PPC guys have caught up to speed very very quickly..... I hate mixing metaphors on the kernel list since they tend to degrade into discussions of social-Darwinism, bunnies, bazookas and wolfs with IT-manager claws (check out the kgdb thread). So I'll avoid the penguin jumping in metaphor. I did take the leap to BK, and took the PPC and RTLinux guys with me. We plowed the path for Linux work with*

[275] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0209.1/1856.html
[276] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0203.1/0027.html

BK already. We've several pitfalls, influenced BK enough to remove some problems (Larry has been really accommodating) and I'd recommend it to nearly anyone. Linux BK doesn't depend on Linus using it. If it did, I wouldn't be using it. We've been tracking Linus for nearly a year, merging with him, taking patches from BK and non-BK users and it does work" (Dougan, 2000 – Sept 14th)[277]. BK requires more learning [or unlearning and then new learning] compared to other VCS like CVS and RCS because it behaves differently and users cannot rely on prior knowledge to guide them. Learning new things and ways is easier if new technology is similar to what has been used before [as it builds on old learning or relearning] and if such problems are dealt with frequently. Level 2 learning is made smoother if levels 0 and 1 have been worked through soundly and with real reflexive understanding. Returning to the idea of learning as a process, Dougan (2000) elaborates on how the use of any VCS guides learning in a particular way. Each stage of learning accomplished will affect all learning that occurs afterwards. One is set on a particular trajectory and to follow it is usually simpler than retracing ones steps [unlearning] and then making other attempts. These new attempts are more knowledgeable as they profit from hindsight learning (Weick 1995, Weick 1996, Weick and Quinn 1999). VCS organizes and as actors use it to organize material they learn through the process of using the software and about the code under production as well. Learning by doing is learning by organizing.

Learning by organizing in action – A clear example of learning and organizing in action is provided by McVoy (2002), "*mirror the tree into CVS/etc and you will very quickly learn why CVS/etc have serious problems. By learning about those problems, you'll either develop some insight which will aid you in making CVS/etc better, and you'll develop a healthy respect for what BitKeeper can do*" (McVoy, 2002 – Mar 5th)[278]. This is learning through organizing *in action.* Using BK files and converting them to use in CVS will cause problems but this process will provide insight about BK to the actor. The actor will either be converted to BK or will be unable to realize the real worth of BK through a lack of prior learning [inexperience] about VCS. The implication is that the actor who truly manages to learn from this experience will understand the superior worth of BK. Thus for the Collective's actors, using a VCS is also about learning through

> "*there's a major learning curve and mind-set change required to work with captured data as opposed to interactive debugging.... Easier for kernel beginners to learn the kernel internals. Having worked on 10+ operating systems over the years, I can testify that some form of kernel/OS tracing facility is extremely useful to get people started. I agree with Linus when he said:*
>
> > *"'Use the Source, Luke, use the Source. Be one with the code.' Think of Luke Skywalker discarding the automatic firing system when closing on the deathstar, and firing the proton torpedo (or whatever) manually. _Then_ do you have the right mindset for fixing kernel bugs."*
>
> *But Linus has also said "The main trick is having 5 years of experience with those pesky oops messages ;-)". Beginners need some way of getting that experience. Reading the source from a cold start is an horrendous learning curve, debuggers help to see what the source is really doing. Always remember that 90%+ of kernel users are beginners, anything that helps to convert somebody from kernel beginner to kernel expert cannot be bad*" (Howell, 2000 – Sept 15th)[279].

Some key points that emerge from Howell's email include; learning as a process implies making mistakes and learning from them; instinctive [Level 0 and sometimes Level 1] learning is natural to all actors; and all actors will take different trajectories of learning for the same problem, thus actors organize material and reality in a way conducive to their own particular learning style and level. VCS makes evolutionary learning for developers so much easier. It makes everything transparent and accessible to all. Each actor can follow the steps taken by others before it and by paying attention to the breakdowns faced by them and what they did to overcome the problem actors are able to learn what to do in particular situations.

Building on past learning – Software creation, especially open source, is in large part dictated by reusing ideas and code of other programs. BK is a better VCS for the Linux collective but BK is the way it is because it has learnt from the mistakes made by RCS, CVS, etc and been able to cultivate the strengths of those programmes, "*Its also not clear that open source companies will replicate everything or have the ability to do so. Even without US patent bogons its doubtful that open source is going to replace Oracle in a hurry. It takes time for stuff to become commodity. As to using one companies lessons to do your work, how much did BK learn from what *didn't* work well in Clearcase. Rather a lot I believe. That learning fuels innovation*" (Cox, 2003 – Apr 27th)[280]. Cox points out that learning fuels innovation as it stimulates ideas for improvement. Technology has to prove that it is better than its competitors and this is only achievable when many use it and understand its benefits. Such translation implies that technology is able to 'speak' through its efficiency and effectiveness, and critical mass acceptance.

[279] http://www.ussg.iu.edu/hypermail/linux/kernel/0009.1/1331.html
[280] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0304.3/0853.html

Learning in Technology

Past learning may strengthen new learning – Technology, like humans, is able to build on old learning, "*note that it's a lot easier to expand tools that already know about revision control than it is to expand tools that have never known about it*" (Torvalds, 2002 – Mar 15th)[281]. Technology needs a strong base upon which ideas and concepts can be built. Learning is evolutionary where some weak ideas are dropped and new stronger ones are accepted which then help technology grow [centre of calculation]. The more intelligent the tool the more capable it is of learning because it makes possible more trajectories of growth.

Technology can be ignorant – The opposite is also true of technology. If it has no prior knowledge of something then it can be ignorant and find learning challenging, "*CVS doesn't even know what a directory is*" (Andersen, 2002 – Mar 7th)[282]. This implies that other forms of technology are capable of 'knowing' and if CVS doesn't know then it is less intelligent software and thus less capable of learning. Developers understand software and how wilful it can be, or simply unintelligent.

Learning can lead to radical change – Learning can lead to radical change in technology as well as humans, "*You can also teach CVS to do this but it would be a pretty big rewrite and you'd end*

implication is that technology is able to learn? And like humans the ability of technology to learn varies from one to another based on its earlier teachings (code, algorithms and design). In order to make CVS understand the needs of the Linux kernel CVS will have to be taught from scratch (plenty of rewriting of the code base so that other trajectories become possible) however, CVS learning will be slower than BK's because BK has already moved on from CVS design and is quite distinct structurally from CVS. Learning happens through rewriting as well, but rewriting is a process of organizing. The collective learns from breakdowns but also learns from the process of trying to fix the problem. Organizing is as much a way of learning as the eureka moment that a breakdown often triggers.

collective to innovate, "*the real difference from BK is that history and tree of changesets are independent things. It's not a "growing tree", it's "changing tree of changesets and its previous forms*" (Viro, 2001 – Dec 27th)[284]. Actors are free to follow any trajectory they want. In this respect BK is more transparent and is also better able to fold time and space.

Collective Learning

Successful collective learning requires critical mass – Collective learning [Level 4] is made possible when enough members of the collective make an effort to learn the same idea. In the Linux kernel an added ingredient makes the latter possible, "*if Linus were willing to dictate from high that we were going to use bitkeeper, and that all patches had to come in as bitkeeper changelogs, then that might get us critical mass. If he doesn't do that, though, my big concern is whether or not it'll be able to garner enough critical mass for it to be worth the trouble for kernel developers to want to spend time learning it*" (Ts'o, 2000 – Sept 13th)[285]. The collective will only take out the time and effort to understand something to learn a concept unless the majority of the collective can be induced to do likewise. This is how collective learning is promoted and how it can be made faster. VCS allows for actors to learn through it as it makes the code visible and flexible but the code written for the VCS itself is also just as open to learning and helping others to learn. VCS allows actors to learn via its use and via the very building and use of the VCS software. It serves a dual purpose or reflects dual learning. All technology learns but how to use it efficiently also requires learning. The more intricate the software often means that the learning of how to use it will be steep as well. The stress in open source is thus on collective learning as the collective always brings the question very pragmatically back to gaining critical mass for any decision to be taken and accepted [translation in action]. Access to the code of others is an asset of the collective because as explained by Ts'o it allows others to view code and allow them to learn from imitation. The collective also helps developers by pooling their resources and creating a learning repository where questions can be asked and answered. All the actors learn from each and develop faster because they can depend on other actors for support and feedback (collective learning).

Individual to collective learning – It is critical mass that allows any collective to make the jump between individual to collective learning,

> "*The real value for bitkeeper comes when I start getting patches from *other* people as changesets, all derived from the same BK "root" repository. Then I can much more easily merge changes, and do all of the things which makes BK so nice. It's this critical mass which is*

[284] http://www.uwsg.indiana.edu/hypermail/linux/kernel/0112.3/0612.html
[285] http://www.ussg.iu.edu/hypermail/linux/kernel/0009.1/1022.html

> *missing...The problem though is time. I took a full day's worth of my time trying to play with BK. That was a significant investment on my part, since I could have done a lot of other things with that time. And in order for me to get really familiar with it, I'll have to spend more time....*
>
> *Now multiply my experience by the several hundred kernel developers out there, and you can easily see how the kernel development community could easily have to invest a man-year or more learning how to use BK. But there's catch-22, in that if we don't have a critical mass of people using it, then the value of BK is seriously diluted. I recall the old story of penguins lined up at the ice floe's edge, each nudging each other to see who would be the first one to jump. Eventually one would get pushed in, and if he/she wasn't eaten by a shark, they would all jump in and they would all get fat and happy. There seems to be nice parallel to this situation*" (Ts'o, 2000 – Sept 14th)[286].

His learning becomes collective learning through gaining critical mass [a stronger network and translation]. Collective learning has a strong imitation flavour to it. Actors learn from each other, the pool of knowledge grows and this ensures widespread adoption and acceptance. If respected members of the collective adopt something then others are more likely to follow, perhaps even blindly because they feel this is safe or more tried and tested an approach.

Collective learning takes time – The implication about garnering a critical mass of actors is that collective learning is bound to happen when the acceptance rate of any technology grows however, very importantly some developers explain that collective learning takes much longer than individual learning,

> "Ted Ts'o: *The kernel development community could easily have to invest a man-year or more learning how to use BK. But If it takes a man-year to learn BK, that person shouldn't be doing kernel work. It's beyond them. Putting shoes on would be a monumental effort of mental power for them.*
>
> Juan Quintela: *I think ted means that it takes a man-year for the kernel community, not for a single person :)))))*" (Quintela, 2000 – Sept 15th)[287].

Actors learn and rely on each other for answers and feedback so it takes longer for the entire collective to learn. Part of collective learning includes the new learning trying to become an OPP. This takes time and effort but if done well, as in the case of BK, it ensures that the entire collective [or a large part] speaks the same 'language' with each other.

[286] http://www.ussg.iu.edu/hypermail/linux/kernel/0009.1/1281.html
[287] http://www.ussg.iu.edu/hypermail/linux/kernel/0009.1/1374.html

Metadata as a source of collective knowledge and learning – Collective learning is often used as an inducement to use BK. The metadata created and held by BK is a vital element needed to understand code written by other actors and it is the circulating of such comments that help create a more fluid learning collective. McVoy insists that BK has been accepted by the collective because it helps to create a learning collective, rather than the fact that Torvalds' acceptance and use of it almost forced some members of the collective to adopt it, "*Linus started using an SCM (source code management) tool called BitKeeper in February of 2002. Since BitKeeper isn't free software, he does not require that anyone else use BitKeeper, he continues to accept patches just like he always did. The only difference is that information about who did what, and maybe why they did it, is recorded and is useful for learning the source base, tracking down bugs, etc. Many, but not all, of the core developers have switched to using BitKeeper because it makes their life easier in various ways*" (McVoy, 2003 – Apr 19th)[288]. Organizing and learning are again coupled

[redacted]

If open source development is 'bazaar-like' then it is given some coherent organizing structure through VCS.

'Becoming in action': VCS makes apparent the grey area of becoming between learning and organizing. This is possible because it allows framing. Framing (Callon, 1998) is the conceptual idea of making all trajectories visible, both successful and unsuccessful ones, which is operationalized through VCS in open source collectives. VCS allows one to study all the various pathways taken by actors and allows us to see what step manifested itself in learning or organizing. VCS is thus both a methodological and practical aid to our better understanding *becoming.*

Learning through organizing: VCS indicates what needs to be organized and how organizing is unfolding. We are made aware of the performative nature of both learning and organizing by following the trajectories of development or even by retracing steps done in the past. VCS organizes and as actors use it to organize material they learn through the process of using the software and about the code under production as well. Learning by doing is learning through organizing. VCS manages the fine balance between keeping slack enough to learn but not too much to cause fragmentation and un-organizing.

Here at the end of the analysis my detailed reading is overwhelmingly one of richness. The data allowed me to explore many intriguing aspects of OS learning and organizing, and sometimes could even be said to have been a real distraction. Chapters 7 and 8 include the main themes which emerged from the data and in the following chapter we set out to answer the research question.

Chapter 9
The Creative Space of Learning and Organizing

In the previous two chapters we have focused on how the Linux kernel collective interpret and use version control software. Now we return to the research question using the empirical data and analysis as a foundation to providing some answers. To recall, the main research question of this study asks:

'*How does learning and organizing occur and manifest itself in open source collectives?*'

The process of both learning and organizing have been explored in this thesis, but on occasion we have artificially kept them separate. However, as mentioned in Chapter 3, the overall position taken throughout the thesis is that it is not appropriate to speak of one without the other, and here we bring them rightfully back together. This idea is not new (Weick and Westley 1996) but has seldom been operationalized in a way that is both theoretically and methodologically powerful.

In the case studied here a combination of dynamics helps create the balance observed between learning and organizing. These are discussed below and include; the LKML; VCS; the license; the politics of space; the centrality of code; and questions of governance style. What brings all of these aspects together and makes them meaningful is their contribution to the establishment of a creative space where changing (becoming) occurs.

Learning and Organizing

Learning needs to become organized to be useful to the collective, and the question of how to better organize and manage the collective will come from better understanding (learning). This claim is evident in the Linux kernel case. Open source development, as we have argued, is a collective process where learning and organizing are revealed in a distinctive way. But organizing is not the same as organized. As explained earlier in the thesis, the focus is not on the achievement of the organized, though the manifestation of organizing does touch upon the former, rather it is on the becoming of organizing itself. For example, to coordinate collective learning certain "movements" (decisions) are made concerning which actor will be invited in to help manage the others. Such moves to improve coordination are acts of organizing. As we have seen, the question of which actor will be asked to help organize the collective is not a minor one, indeed it is a highly political one with great significance for the survival of the whole collective. Instead of (or alongside) Linux kernel code we see VCS code become the larger matter of concern around which other actors assemble and hold serious discourse. This we learn is because VCS is designed in a certain way and aligned to certain interests, which in turn affects how the space where actors assemble is structured and how actors and information move inside it. In other words the very Constitution of Linux kernel development is under debate.

In this study we have approached this issue empirically through the LKML, and made discussion of VCS the focus. VCS, in Clegg *et al.*'s phrase, is 'the knot, the fold, where order and disorder meet' (Clegg *et al.* 2005). Patches and developers meet in this space to build attachments and their own network, and each actor is attempting to entice and command others. VCS (the idea of VCS) is more than a mere conventional meeting point for different actors; it is (potentially) a new *creative space* that can allow collective learning and thus organizing of the collective, through its own particular form of transparent and highly political governance. This is a creative space that can be folded by actors, but without depriving other actors of the ability to do the same for themselves. It is also a space whose traces are archived and can keep the collective fluid and flexible.

Creative Space

As suggested above, we see VCS as a particular kind of space, a 'creative space', where in Latour's (2005a) words various matters of fact come together and struggle to become matters of concern. As various actors point out, VCS allows one to follow the trajectory of patches and bugs, and observe collisions and fusions. We have also seen that this is a political space where patches struggle to gain visibility as VCS allows each patch to show off its ability and strengths. The visibility afforded makes possible learning and innovation by imitation, and adoption. Of course, VCS does not act alone to build a creative space, the LKML shares, to some degree this ability. Latour (2005a, p6) would call this space the Body Politik, "*How to represent, and through which medium, the sites where people* (the Body Politik is composed of humans, material actors, and hybrids) *meet to discuss their matters of concern? It's precisely what we are tackling here...: How do they assemble, and around which matters of concern?*".

The extent of visibility that any Body Politik is willing to extend is given in large part, in open source and other software processes, by the licence (implicit or explicit) under which it is created, where the more 'free' the licence the greater the transparency of actors and their movements and vice versa. Here we see an interesting dilemma emerge as the GPL aligned software is forced to work closely with closed source software. This casts both short-term and long-term doubts on the visibility of code and metadata, and thus on the democracy of the Body Politik. Indeed the Constitution of the kernel is seen by some to be threatened. This issue of transparency (openness) is central to open source development, seen by most as fundamental to ensuring the survival of the code. The alien VCS, BK, is considered to be at best 'ajar source' and this brings up the possible hybrid nature of other code in the collective. The GPL has no room for mixing with code of other licences as it automatically converts any code into GPL'ed software. Yet we see emerge a trend towards hybridization in open source projects, including Linux kernel development, for a time[289]. But we also see that code, amongst other actors, is too pragmatic to not make use of differently licensed allies. Whatever is created is also then necessarily of a hybrid nature. The coexistence of opposing actors in the same space changes the nature of the creative space and how learning can move (becomes, organizes) within it.

The need for coordination and organizing emerges as a network grows and becomes more interconnected with other networks. It is usually at this point that decisions begin to be processed (becoming), and choice of fidelity to the licence is one of them – questions of the licence offer a guiding set of rules for current and future organizing, including what the code is able to learn and to teach (allowed to do). From this perspective the GPL guides the development process and, broadly speaking, decides what can be done with the code and what is not possible – the Constitution (Weber 2004). As the GPL organizes the boundaries of what code can do, it is the associated but not equivalent governance model, no matter how emergent, which "organizes" the developers. We then argue that both the license and associated governance model structure the creative space, which in turn makes some forms of learning possible while restricting others.

What we also see from the case study is that the governance changed over time, beginning with a clear leader, Torvalds, to whom all other actors would look to make decisions on code, process, releases etc. However as the network grew so did his responsibilities. Organizing kernel development was becoming unmanageable so some work was redistributed to other actors like Dave Miller and Alan Cox (trusted lieutenants). The success of the kernel continued which attracted yet more actors to the network. The level of learning and number of learners that required organizing grew quickly. We saw a technical actor come forward to facilitate the sorting (Bowker and Star 1999) of issues but this actor, CVS, was viewed with great mistrust by the leader. This spurred a drive for new organizing, expressed as a search for the appropriate VCS. For various reasons the VCS that emerged from this desire, BK, was not welcomed by the collective wholeheartedly, indeed it was subsequently rejected.

[289] Linux kernel developers used BK for a number of years before McVoy pulled his VCS away from this collective because he felt some members were trying to reverse engineer his closed source software. It was at this point that Torvalds decided to create his own customized, open source VCS – GIT.

We argue here also that material actors too are capable of their own agency in learning and organizing. This claim is premised on the view that all actors of a network possess agency to act, or discover how to acquire it to achieve their goals. In the two previous chapters we have seen various instances of material agency. For example how CVS was seen to threaten to become more powerful than another actor (Torvalds). To achieve this CVS gained an assemblage which was beginning to match Torvalds' following and CVS was able to engineer such growth *because* it gathered the requisite agency from other actors. The findings of this research thus corroborate the constructive-realist ontology of ANT. We see material actors harness agency from human actors, and once they are able to do that, how they behave is as unpredictable as human actors with agency. Agency is fluid and mobile and we see it dance between actors thus giving them the capability to act and attempt to gain critical mass for their assemblage. Furthermore, material actors, like human actors, need to change and constantly evolve to keep other actors interested. Such change, we argue, is a manifestation of learning and can support constant improvement in the collective, fed back into the collective through organizing.

VCS organizes time and space, patches, metadata, different branches of development and developers. In the previous chapter we elaborated on various examples of collective organizing through VCS use. One example often repeated by actors is the role VCS plays in making the organizing of learning transparent. Actors can view collisions and fusions and can thereby, better understand (learn) which trajectories of code are more appropriate. A closely linked issue is that of organizing through branching. VCS allows actors to take any idea and organize it further in another trajectory if they so wish. Each trajectory is part of the overall becoming of open source development because it helps offer the potential to differentiate one segment of code from another or reformat code in a new way.

VCS organizes patches and developers through branching but also via the merging facility. As patches are sent into the VCS by various actors across time and space it is the role of the VCS to make sure there are no conflicts and if any do arise then it must resolve them by either flagging them up or, in the case of BK, by its auto-merge organizing ability. The merging technique is an expression of organizing as patches are sorted and the question of why one is better and thus acceptable to the collective is resolved. Once merged and accepted the patch becomes part of the larger code base (learning manifested) stored in the code repository.

Becoming Through the Creative Space

The ability of VCS to frame a creative space for the collective is premised on its power to make transparent the *becoming of learning and organizing in action.* Becoming, as explained in Chapter 3, is an essential space of connections that express the dynamics of learning and organizing. Becoming is not the specific state that is left, nor what is achieved, but the movements that take us from 'then to now', or now towards some future, embodied in the tension between outward looking and exploratory processes of learning, and cautious and considered acts of organizing. This duality has to coexist in order for either to operate, a balance that Clegg *et al.* (2005) describe as a 'generative dance on the edge of a volcano'. Becoming is then a way of naming and revealing the space where learning and organizing resolve and shape the futures of the collective. Remember, we understand VCS as making possible more than one future or trajectory through its ability to branch and fold time and space, and thus there is always, and often in parallel, more than one way that organizing or learning is taking place.

The fleeting or transitory nature of becoming makes it challenging to study since the instant you capture it you realize that you have focused at best on some short-lived stability. For this reason we have coupled the idea of becoming with ANT's concept of framing to make becoming more visible but less frozen. Ontologically, both concepts are compatible because both stress the creation of reality in real-time. Thus reality is performed and unfolds and emerges, it is processual. Framing adds a dimension that helps operationalize becoming. Framing is the process of making visible all the transactions, and relationships that actors experience, and significantly, these include all the failed relationships and transactions too (Callon 1998b). Framing thus makes learning and organizing more visible, "these investments apply and produce both knowledge, in that they cause hitherto invisible links to appear, and also a reconfigured collective in which these now visible and calculable links have been renegotiated" (Callon 1998a, pp259-260). The process of learning inspires actors to traverse varied paths before they achieve the momentary stability that our research reveals. Often it is the abandoned paths that become the catalyst for learning and so we argue that a researcher needs to follow an actor through both successful and unsuccessful paths to disentangle the process of learning and organizing the actor participates in. Framing can in this way operationalize becoming.

Use the Source, Luke, use the Source. Be One with the Code[290].

Framing gives us the ability to locate, trace or follow interactions, but in open source what exactly is it that we are following? The final claim made here is that it is the code that must be followed because in the end it is the code that learns and organizes. Code as an organizer of learning is the concept that allows us to move from a macro perspective of creative space to the more micro one of becoming in action. In this study we show source code to be the primary learning and organizing stratagem. It is also the way learning travels from the individual actor to others to create collective learning. It is the most precious resource of the collective and remains the main motivation for actors to assemble together. We then argue that code is a strong actor with agency and the ability to harness more, to force and insist that the collective become pragmatic, even in the face of a direct clash of constitutions of licences.

Code we argue brings the collective together and creates around itself a network of actors willing

[redacted]

mutable mobile which circulates and spreads learning across the collective (and is what makes possible boundary crossing between different collectives). Code gathers ideas in its movement through the collective and interacts with other actors. Through this interaction the code passes or delegates ideas and concepts. Delegation, in actor network theory, is explained as shifting in or out (Latour 1987). Shifting implies movement in time and space. Centres of calculation, of which VCS is a prime example, are the accumulation of many traces where time and space can be (re-)constructed locally.

Code is then the strongest actor in the collective, and it has its spokesmen, e.g. Torvalds. If Torvalds decides that "his" code needs a closed source VCS then contentious issues of constitution differences are swept aside to give code what it needs. Code is pragmatic and encourages the collective to be likewise, this in spite of the collective's strong ideology that suggests otherwise. The collective begins with a drive to create and maintain code and in effect is driven by the same throughout its performance. And if code is not convincing an open source project will wither and die (as indeed most do!).

Manifestation of Learning and Organizing in OS Collectives

This research suggests that nothing is stable in open source and we do not use the word manifest [292]or manifestation in the main question to mean something static or unchanging. We refer rather to the traces of learning and organizing found in the becoming. The significance of manifesting is explained below through a number of chosen examples from the study.

One of the key forms of learning's manifestation in most open source collectives is the development mailing list, for example the LKML. Any email message sent to the group email is ruthlessly archived on various mirror sites[293] for both current and future reference. The archival nature of such sites is enhanced with a proper search facility and a clear breakdown of information by subject, date and even author. The findings of this study show that past emails are often referred too by various actors when attempting to make an argument. Such sites are a key resource because this is the assembly point (Latour 2005a) for various actors such as developers, and patches (as they are often emailed to the collective at large), where discussions are held about the merits of patches, decisions taken and possible new approaches to development proposed. The openness of the mailing list ensures that all actors have equal access to it to view and read, but

[292] Manifest means to 'display (a quality, condition, feeling, etc.) by action or behaviour; to give evidence of possessing' (OED) and that is how it is understood in this research, or perhaps we would be better saying manifesting as all forms of learning are continuously being changed.
[293] A number of such sites were discussed in Chapter 5.

also to contribute, thus adding to the "democratic" nature of open source development – actors are given some potential voice through the LKML – themselves or via spokesmen.

The LKML is thus not only a manifestation of learning, it is a manifestation of learning *being organized*, LKML and its continued use by the collective indicates a manifestation of organizing. Over time the LKML has changed with various forms, like the decision to organize the email messages first by month only but then as the need for greater organizing arose (due to an increased number of emails exchanged) by week as well and then the addition of search interfaces.

Another manifestation of learning in the Linux kernel collective, we argue, is the archive of code which is stored and accessible via the version control software used. The patches and code are a manifestation of collective learning indeed the most precious resource of the collective, not just its output. We see that every attempt is made to collect such a learning repository from the long email exchange about the 'kernel release history' (see Chapter 6). Prior to the use of any accepted VCS the Linux kernel collective were forced to keep numerous personal archives which were often incomplete. This became a growing worry to the collective and was part of the reason that spurred the debate over VCS adoption. Actors express a need to access any segment, patch, or trajectory that they wish at any time or place in other words to be able to fold time and space in a seamless manner. VCS can do this and it provides a window for actors to 'see' breakdowns' and collisions happen as the catalysts for learning, explained in Chapter 8 as the transparency of control.

Each release or version of software is the forced or artificial manifestation of organizing. Actors contrive a stability point in the code and it is released, but again this is no indication that there will be no more changes. Stability is a pretence because the collective releases code but that doesn't preclude actors from adding to and changing it. Even the released 'stable' versions can and sometimes are taken up by other collectives who then begin to reframe the code in a way that suits them. In this way the parallel release strategy in Linux kernel development is a good example of the manifestation of organizing. It is not stagnant, it does change and evolve, and suggestions are often made to increase the number of concurrent releases, but at the same time it reflects momentary stability in the collective where a consensus has been reached by actors and so for some moments there is stability. However this stability is of the sort that Bateson illustrates well when he describes the trapeze artist walking a tightrope. The artist appears not to be moving at all but in fact his/her muscles are working frantically to appear steady.

VCS holds patches and code but VCS also create and hold metadata (code about code). As mentioned earlier metadata are comments and notes added by patch authors that help make the submitted patch understandable (including author details and timestamp). These are generally as essential as the patch and we see this in the serious altercation between different actors when the metadata is threatened by BK. Metadata explains the reasoning of patches and allows other actors to follow traces of kernel development. Again, like the VCS, metadata helps make sense of not only the patches but also the breakdowns that have occurred, and helps to explain the steps taken to overcome a crisis.

The final conclusion is then that the most significant manifestation of learning and organizing in open source is the code. That is code is what makes learning happen and *is* learning. It passes from actor to actor in the network showing off its abilities and at the same time inspiring others to make changes to it to improve it yet further. This builds on our argument that code has more than one function. Its relevance in the collective is suggested by the panic its potential loss causes and the real effort that is put into maintaining and sustaining a repository (archives) of it for the future.

Code creates a bridge between individual to collective learning in open source because code is mobile and promiscuous. Under the GPL every patch created is owned by the author but the GPL allows that anyone else can then use, read and improve that patch which creates an impression of collective ownership (but we see it differently). Collective behaviour is learnt and circulated through open access to the LKML, metadata, learning repository and the code. A different licence would change the dynamics of what and how code travels and who can access it. The GPL doesn't allow mixing of licences but the recent trend in open source projects has become dual licensing where two or more different licences are adopted for the code developed. Sometimes this can lead to a hybrid process, business model and perhaps even code, though the latter is not evident from the current study.

Arguments in Summary

This chapter has explored our findings in relation to learning and organizing as a performative duality. From this we draw an understanding of how open source collectives learn to organize and organize to learn better. The study of *learning through organizing* based on the concept of becoming, a concept we couple with Callon's idea of framing to help make transparent the interactions between actors which are the performance of learning and organizing. The canvas upon which this becoming is revealed in this study is a creative space, a VCS. This space is structured, governed and moulded to create and reinforce certain power structures. Code and other actors move within and, in this creative space, changing and organizing others. We conclude that in open source collectives it is essentially code that is the manifestation of learning, but also the means of organizing. This may appear to be a rather technologically deterministic explanation but our understanding of code is not merely as a technical artefact. Rather, code is a political, social and technical hybrid which aggressively and selfishly manipulates the collective to prolong its survival.

Can we 'Generalize' from these Findings?

The word generalize is often associated with positivist research, and the current work does not fall within that category. So how do we indicate the relevance of interpretivist work beyond the area we have researched? Lee and Baskerville (2003) argue that we cannot generalize from any type of research if we are trying to generalize *across* different cases. They quote Geertz (1973) who encourages "not to generalize across cases but to generalize within them" (pp25-26). Positivism aims to achieve, through research, some universal laws but Lee and Baskerville apply Hume's (1978) truism of induction to prove that the very process and logic of induction is flawed and thus invalid (Rosenberg 1993). Hume's main criticism of induction is summarized by Campbell and Stanley (1963, p17 - quoted within Lee and Baskerville, 2003),

> "The problems are painful because of a recurrent reluctance to accept Hume's truism that *induction or generalization is never fully justified logically*. Whereas the problems of *internal* validity are solvable within the limits of the logic of probability of statistics, the problems of external validity are not logically solvable in any neat, conclusive way. Generalization always turns out to involve extrapolation into a realm not represented in one's sample. Such extrapolation is made by *assuming* one knows the relevant laws. Thus, if one has an internally valid (design), one has demonstrated the effect only for those specific conditions which the experimental and control group have in common... *Logically*, we cannot generalize beyond these limits; i.e., we cannot generalize at all. But we do attempt generalization by guessing at laws and checking out some of these generalizations in other equally specific but different conditions. "

Based on Hume's premise Lee and Baskerville (2003) build a framework of four types of generalizability. Suffice it to say here that they understand most interpretive research to fall within the second category of their framework which explains generalizing from description (data) to theory. Again it is stressed that such generalizations can only be made within the data collected and not across case studies. Generalizing from description to theory implies a greater understanding of the theory through a study of the data. It is a way of adding to where the theoretical framework proved inadequate or broke down. In this spirit the next section is my personal experience of a breakdown in the theoretical framework used and how my findings helped to build on both theory and my understanding of the findings to create a concept which is relevant to other researchers and fields (within the remit of Hume).

Thus while we agree with Lee and Baskerville's (2003) point concerning not being able to generalize across case studies yet at the same time our understanding through this study has encouraged us to discern some implications, for theory, practice and methodology, which could have relevance beyond this study.

Theoretical Discussion and Contribution

This study of the Linux kernel collective has enabled a better understanding of the constructive-realist ontology of ANT. For this researcher the point of access to Linux kernel development was via the LKML which made it possible to follow reality creation being performed. The constructive-realist ontology of ANT is premised on the claim that both human and non-human actors are able to act – they have agency or are able to acquire it. Non-humans acquire agency from other actors, but once acquired, what the non-humans are capable of doing with it is unpredictable and in many cases innovative and evolving. We see this in the case of the collective's adoption and use of BK, and how once BK acquired enough agency it was able to make itself (for a time) indispensable, grow in power and control over the network, and to coerce the collective to help to evolve and learn. Agency dances between actors and is what provides actors with the ability to control other actors and become meaningful within the network. Both human and non-human actors need to evolve and change in order to remain meaningful and in control. This they achieve through agency which they can galvanize to change, evolve, and to learn. The study indicates just how significant continual learning is to sustaining the collective. Non-human learning rests on obtaining agency with which to control and direct the future of code and the collective.

The research framework created with ANT coupled with Bateson's definition of learning and Weick's ideas of organizing showed code to be the main *learning and organizing stratagem* in open source collectives. Learning and organizing are strongly linked but more importantly it helped to build on ANT theoretically and methodologically because linking organizing with learning (inscription) gives impetus to the circulating reference as more than just learning but also organizing. Inscription is organizing of learning or learning *through* organizing as explained by ANT. ANT concepts coupled with Weick's ideas of organizing help operationalize both learning and organizing. ANT complemented by Bateson's learning and levels of learning adds depth to the former because ANT does not make a clear distinction between changing and growing. As mentioned in Chapter 7 growing is usually defined as a gradual increase in something but change or changing is a more radical transmutation. ANT mentions change and growth but never quite makes a clear distinction. The work of Law and Mol (Mol and Law 1994, Moser and Law 2006) on fluid objects, when linked to inscriptions and centres of calculation, helps to develop ideas of learning as change, yet it would still be rather difficult to distinguish between various levels of learning through just an ANT lens. Bateson's levels of learning then lend an added dimension to our understanding of change and we are able to distinguish between different types of learning. This is one way to counter critiques of ANT that complain about the 'flatness' of reality when understood through ANT. The levels of learning concept enables us to understand unlearning and adds to ANT which is unable to explain properly level two or three learning where the idea is to learn to learn.

ANT's concept of framing is powerful and when coupled with Chia's (1999, Tsoukas and Chia 2002) ideas is an effective way to operationalize becoming. Chia speaks of learning and organizing and the grey area between which he calls becoming. It is in this creative space that 'something happens' to create learning or organizing. Framing operationalizes becoming, or in other words it helps make available the performance of 'becoming in action'. ANT is accused of allowing a researcher to only follow successful trajectories at the expense of other possible becomings. The concept of framing (becoming) mitigates the criticism of ANT that it only allows the macro (strong) actor to speak, or be heard – this is keenly obvious in the Linux kernel case study.

Time plays an implicit role in ANT and there has been little work to push this area further except by Kavanagh and Araujo (1995) on time and space folding (they call it chronigami). We claim here, as do Kavanagh and Araujo that time is processual (performative) but at the same time can be folded and accessed at any point. The current study is able to operationalize their concept. We also add that the 'chronigamic' ability of VCS makes for a more effective method of organizing and learning diffusion. It allows for more creative action as actors are able to work on any thread of development that interests them. This helps create a balance between slack and stability. It also adds to science and technology studies [STS] and actor network theory as applied to an open source case which not only helps to understand the case better but also leads to further insight into the applicability and limits of this theory and school of thought.

Contribution to Practice – Open Source

As a social scientist the contribution to the technical understanding of open source and version control software may be slim, but this has not been the aim of this research. Instead this work has led to a better understanding of the social, political, and even economical issues concerning VCS adoption. VCS has been and is used in open source development but is usually taken as matter of fact (Latour 2004b, Latour 2005a) rather than the matter of concern that this study makes evident. A matter of concern (Latour 2004b) is a divisive and problematic idea, it is 'disputable' (Latour 2005c). Yet in most literature VCS is accepted as an unproblematic and apolitical tool to be used by open source collectives. This study made obvious the contestable nature of VCS and this can help to make sense of the role other 'tools' are also capable of playing and how such technology should be viewed warily (Ciborra 1999).

Technical actors are political actors and need to be understood as such. VCS plays a vital role in open source development and we recognize from this research that the design of space of VCS is closely linked to how the collective is governed. It is a political creative space that can make the flow of information transparent to some while restricting others, thus it organizes space to match the governance mode and vice versa. The mode of governance is also framed by the adopted license. A better understanding of VCS and other software is a way this researcher would like to contribute back to the open source collective she has studied over the last few years.

VCS plays an integral part in encouraging, organizing and distributing learning. Repositories or archives (organizational memory) that are made possible through VCS spur collective innovation and learning. Our findings indicate that the actors of the Linux kernel development collective place great value on not only old code, patches, and metadata but also archive their discourse [the LKML]. They actively tried to assemble a history of Linux where all previous code written was organized. Individual actors in the collective, especially the more experienced are sensitive to the need for an archive but this research helps understand in more depth the need for an up-to-date archive of all communication, of code or about it. In part, perhaps because open source developers are keen to reuse rather than reinvent but the strength of their need to protect this knowledge base shows a deeper and long term need to preserve the code as teaching and learning material to support novices and create core programmers. In other words an archive is needed for survival of the collective.

The archive needs to include metadata. Torvalds made a mistake when he didn't accept McVoy's offer to allow metadata to be GPL'd along with kernel code in October, 2002 and McVoy revoked his offer in early 2003. The current study makes more apparent the role metadata plays in learning and its diffusion in open source collectives. The Linux kernel code is difficult to understand without metadata and this study indicates how both the code and metadata need to be under the same licence for the collective to flourish. Code may appear pragmatic when a license concern crops up but it seems that metadata is not able to move as freely. License and regulations are able to restrict metadata so open source collectives need to be more careful about keeping ownership of metadata within their control.

In open source it is code that is the *learning and organizing stratagem* – code is both the carrier and catalyst for learning and organizing. Open source developers are aware of the significance of code but perhaps not so clear on the vital scope code plays in sustaining the collective (as described above). The crisis faced by the Linux kernel collective of closed source software making attempts to trespass on open source software provided insight to the actors concerning the depth to which their process and product could be affected by a mixing of constitutions (licenses). This crisis lead to ideological conflicts, ownership problems, reusability concerns, question of distribution styles, and a discussion of which licence should prevail.

Lessons for Open Source

We summarize here the main points from this thesis that could be pertinent for open source developers.

- Software, like hardware, has an architecture that organizes space in a specific way. The architecture of software is both a way to create openness in the collective but at the same time it is restrictive, depending on the level of access an actor is allowed. Thus developers need to be aware of the political spatial issues they create when certain software is designed and then appropriated for use.
- Research on open source software has largely ignored understanding software as anything other than a technical artefact. This view, though important, blinds one to the political manoeuvrings that software is capable of which this study makes very evident. A greater awareness of the capability of software to behave in a fashion not intended by the developer and the possible repercussions of this not simply on functionality but on the political element of organizing is needed.
- The idea of openness has layers which are reflected in the licence for software. The strong drive in open source towards pragmatic solutions and a desire to blend into the mainstream is creating a hybrid form of software, process, licence, and collective. What sort of future does this trend spell for open source? This thesis raises such questions through the crisis of metadata loss and a need for the collective to press forward with Linux kernel software work despite the obvious problems encountered with BK use.
- A more detailed understanding of version control software from a social science perspective makes visible its distinctive and sometimes subtle role in how learning and organizing occurs in distributed development collectives. A study of the open source

process and how temporal and spatial issues are managed in distributed development through the use of software creates implications for areas other than software. There is much to study and learn in the open source model of collaborative learning and organizing but our sense-making of their process can then be pooled back to open source. We believe that this particular study of how version control software allows learning and organizing to occur at various levels through sustaining the assemblage, control over time and space, and the ability to act and make a difference all contribute to how learning and organizing unfolds and is manifested.

- Different governance structures adopted by open source projects are capable of creating environments that are on a continuum ranging from very open to the more dictatorial. The Linux kernel collective study provides insight into how a benevolent dictatorship model develops and unfolds under crisis situations. This study reflects the strengths of such a model when a single decision maker is needed to draw the project out of a difficult situation but also shows how the open source way (or one manner of it) causes a breakdown when the leader finds it impossible to scale. Burdened leaders often adopt software to lessen their load sometimes without reflecting on the consequences of such a move on the hierarchy within the collective and the power struggles that ensue (even between the leader and technology).

Methodological Discussion and Contribution

This study shows that online data, data collection and analysis can offer a very rich data set to study a collective (about how actors in any field communicate, live, organize, learn etc). Online data can be seen as true a representation of a real world situation, as true as other forms of collected data (from interviews or ethnographic observations). Online data collection and analysis provides a more accessible route and methodology to study distributed online collectives such as open source (and other forms of online activity).

In this study we operationalized Law's (1994, 2004) concept of organizing narratives. VCS makes parallel dimensions possible and this draws an interesting parallel with Law's organizing narratives and Mol's (2002) multiple realities concept (multiple realities out of which 'one' reality emerges as stronger). Textual analysis of the collected data not only provided this study with a more rigorous method of analyzing data but at the same time helped to create and make visible the links between various narratives so that 'one reality' (the researcher's mode of ordering) could emerge. This helped to operationalize our ANT methodology.

Operationalizing Law's ideas contributes to traditional research methodology which seems a little slow to catch up with research conducted via the Internet. Most researchers that study email archives or any such large repository accessible through the Internet are faced with the problem of data mining and then analysis of large amounts of information. Some examples of narrative analysis in IS exist (Alvarez and Urla 2002, Davidson and Chiasson 2005) but there is no real tradition of such a methodology in the IS field (though medicine, law, and especially sociology have long been using similar methods). There is also some cynicism about how traditional methods of research have so far been adapted to Internet research. The method used in this research we have termed as an *archival longitudinal study*. This was a longitudinal study of the Linux kernel collective and spanned over eight years of the case yet as it was done online through a study of the LKML archive we were able to do the study and collect the data in a much 'shorter' time. Longitudinal case data collection can thus be done in a few months if the data is online and archived. This contributes to an understanding of what it means to do a longitudinal study. However, the data analysis time can (and was in this case) quite time-consuming.

We kept a diary of the steps taken during data collection and data analysis and this material was then used in writing the methodology (Chapter 5). Our detailed explanation of our archival longitudinal study shows the rigorous manner in which every step was executed and can thus serve as a possible example for other researchers interested in a similar methodology. To conclude, the IS field has few studies that focus on data which has only been collected from online sources but the Internet and all it has to offer is an area that needs to be tapped for accessible and straightforward data collection. As we point out above accessible data collection does not imply that data analysis is any easier than when other methods are adopted but we hope that our meticulous account will be a guide to others wishing to pursue an archival longitudinal study.

Future Direction and Research

There are many possible and interesting avenues which this researcher would like to develop as a follow-up study to the current work. Some are related directly to this work and others reflect a need to be able to use the current understanding to shed light on areas other than software.

A study which deeply interests this researcher involves researching the impact of a fork in a collective on learning and organizing (including the emergent governance structures). The current study focuses on a high profile open source case but one could do a comparative study through an investigation of a small OS project to better understand how learning and organizing occurs in the average small OS collective. It would also be fascinating to do a follow-up study on the Linux kernel collective to trace the progress of GIT and its interaction with the collective. How different an actor is GIT from say CVS or BK and compare the various VCS on learning diffusion and governance. Another exciting study involves a focus on the trend towards hybridization in OS and with OS adoption. There is hybridization of business models, license, code (perhaps) and governance, but what are the implications of this trend?

Taking a more broad perspective of OS ideology it is necessary to see the application of OS principles, practices, ideology and techniques to areas other than software. Some examples of areas that could find the OS way useful include education, and health, to name a few. This would have implications in both developing and developed countries. One could also explore the impact and relevance of the OS model on various loosely coupled 'organizations' spread across time and space in the world. The form of communication adopted by such virtual forms of organizing could also be studied to appreciate how effectively communication occurs and what purpose besides simple interaction do communication methods serve in collectives.

Reflecting back on this study the researcher can say that there is much more to learn, and open source as an area and phenomenon still has much to offer research-wise. This research has been a performative experience for the researcher which was extremely enjoyable. She made many choices for different reasons which led her to traverse certain trajectories. These choices are justified at every stage but this is not to deny that approaches adopted by other researchers are not useful – this study simply offers an alternative.

References

Aigrain, P. (2001) *Positive intellectual rights and information exchanges,* MIT http://opensource.mit.edu/papers/aigrain.pdf.

Aigrain, P. (2002) *A framework for understanding the impact of GPL copylefting vs. non copylefting licenses,* MIT http://opensource.mit.edu/papers/aigrain2.pdf.

Akrich, M. (1992) "The De-scription of Technical Objects" in *Shaping Technology/Building Society*, (Bijker, W. E. and J. Law eds) MIT Press, Cambridge, MA.

Akrich, M. and B. Latour (1992) "A summary of a convenient vocabulary for the semiotics of human and non-human assemblies" in *Shaping technology / building society: studies in sociotechnical change*, (Bijker, W. E. and J. Law eds) The MIT Press, Cambridge, Ma, pp. 259-264.

Alavi, M. and A. Tiwana (2003) "Knowledge Management: The Information Technology Dimension" in *The Blackwell Handbook of Organizational Learning and Knowledge Management*, (Easterby-Smith, M. and M. Lyles eds) Blackwell Publishing, Oxford, pp. 104-121.

Alvarez, R. and J. Urla (2002) "Tell me a good story: using narrative analysis to examine information requirements interviews during an ERP Implementation", *DATA BASE for Advances in Information Systems,* **33 (1),** p. 38–52.

Amsterdamska, O. (1990) "Surely you are joking, Monsieur Latour!" *Science, Technology and*

KernelTrap.org

e http://libre.act-

rsity of Indiana

Open Source 3D

ring Knowledge,

nal, Jossey-Bass,

pective, Addison-

tion perspective,

tion Perspective,

nce, **86** pp. 478-

in Open Source

Andrews, J. and L. McVoy (2002) *Interview: Larry McVoy,* http://kerneltrap.org/node.php?id=222.

Anonymous (2005) *The Libre Site for Free Software Developers,* AdaCor europe.fr/.

Anvin, H. P. (2004 - Thu Jul 22) *Re: linux-kernel CVS gateway?,* Unive http://www.uwsg.indiana.edu/hypermail/linux/kernel/0407.2/0490.html.

Aoki, A., *et al.* (2001) "A case study of the evolution of Jun: an object oriented multimedia library." in *23rd ICSE Conference,* Toronto,

Argote, L. (1999) *Organizational Learning. Creating, Retaining and Transfer* Kluwer Academic Publishers, Norwell, MA.

Argyris, C. (1982) *Reasoning, learning, and action: Individual and organizatio* San Francisco.

Argyris, C. and D. Schön (1978) *Organization learning: A theory of action pers* Wesley, Reading, Mass.

Argyris, C. and D. Schön (1996) *Organization learning II: A theory of ac* Addison-Wesley, Reading, Mass.

Argyris, C. and D. A. Schon (1978) *Organizational Learning: A Theory of Ac* Addison-Wesley, M A.

Ashby, W. R. (1940) "Adaptiveness and Equilibrium", *Journal of Mental Scie* 484.

Asklund, U. and L. Bendix (2002) "A Study of Configuration Management Software Projects", *IEE Proceedings - Software,* **149 (1),** pp. 40-46.

Atkins, D. L., T. Ball, T. L. Graves and A. Mockus (2002) "Using version control data to evaluate the impact of software tools: A case study of the Version Editor", *IEEE Transaction Software Engineering,* **28 (7),** pp. 625-637.

Attewell, P. (1992) "Technology Diffusion and Organizational Learning: The Case of Business Computing", *Organization Science,* **3 (1),** pp. 1-19.

Augustin, L. (2003) *Open Source Workshop - Portland, Oregon* (Personal communication).

Barr, J. and L. McVoy (2003) *Larry McVoy on BitKeeper, kernel development, Linus Torvalds & Bruce Perens,* LinuxWorld http://www.linuxworld.com/story/32618_4.htm.

Bateson, G. (1963) "Exchange of information about patterns of human behavior" in *Information storage and neural control,* (Fields, W. S. and W. Abbott eds) Thomas Books, Springfield, IL, p. 173–186.

Bateson, G. (1972) *Steps to an ecology of mind,* Chandler, New York.

Benkler, Y. (2002) "Coase's Penguin, or, Linux and the Nature of the Firm", *Yale Law Journal,* **112 (3),** pp. 369-446.

Berger, P. and T. Luckmann (1966) *The Social Construction of Reality,* Doubleday, New York.

Berglund, M. (2000 - Mon Sep 11) *[ANNOUNCE] Darkstar Development Project,* University of Indiana http://www.uwsg.iu.edu/hypermail/linux/kernel/0009.1/0472.html.

Bergquist, M. and J. Ljungberg (2000) "Communication: From Management to Organization" in *Planet Internet,* (Braa, K., C. Sorensen and B. Dahlbom eds) Studentlitteratur, Lund, Sweden.

Bergquist, M. and J. Ljungberg (2001) "The Power of Gifts: Organising Social Relationships in Open Source Communities", *Information Systems Journal,* **11 (4),** pp. 305-320.

Berliner, B. (1990) "CVS II: Parallelizing Software Development". in *Proceedings of the USENIX Winter 1990 Technical Conference,* Washington D.C.,

Bettis, R. and C. K. Prahalad (1995) "The Dominant Logic: Retrospective and Extension", *Strategic Management Journal,* **16** pp. 5-14.

Bezroukov, N. (1999) "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)", *FirstMonday: Peer Reviewed Journal on the Internet,* **4 (10),** pp. 1-23.

Bijker, W., T. Hughes and T. Pinch (1987) *The Social Construction of Technological Systems,* MIT Press, Cambridge, MA.

Bijker, W. E. (1993) "Do Not Despair: There is Life After Constructivism", *Science, Technology and Human Values,* **18 (1),** pp. 113-138.

Bijker, W. E. and J. Law (1992) *Shaping Technology/Building Society: Studies in Sociotechnical Change,* MIT Press, Cambridge, MA.

Bittman, M., R. Roos and G. M. Kapfhammer (2001) "Creating a Free, Dependable Software Engineering Environment for Building Java Applications". in *23rd International Conference on Software Engineering: First Workshop on Open-Source Software Engineering,* Toronto, Canada,May 15, 2001, IEE.

Bitzer, J., W. Schrettl and P. J. H. Schröder (2005) *Intrinsic Motivation in Open Source Software Development,* Idea: EconWPA http://ideas.repec.org/p/wpa/wuwpdc/0505007.html.

Bloor, D. (1976) *Knowledge and social imagery,* Routledge & Kegan Paul, London.

Boland, R. J. and R. V. Tenkasi (1995) "Perspective Making, Perspective Taking", *Organization Science,* **6** pp. 350-72.

Boland, R. J., R. V. Tenkasi and D. Te'eni (1994) "Designing information technology to support distributed cognition." *Organizational Science,* **5 (3),** pp. 456-475.

Bolinger, D. and T. Bronson (1995) *Applying RCS and SCCS: From Source Control to Project Control,* O'Reilly & Associates,

Bonaccorsi, A., S. Giannangeli and C. Rossi (2004) "Adaptive Entry strategies under dominant standards. Hybrid business models in the Open Source software industry."

Bonaccorsi, A. and C. Rossi (2003) "Why Open Source software can succeed", *Research Policy,* **32 (7),** pp. 1243-1258.

Bonaccorsi, A. and C. Rossi (2004) "Altruistic individuals, selfish firms? The structure of motivation in Open Source software", *First Monday, Peer Reviewed Journal on the Internet,* **9 (1).**

Bowker, G. and S. L. Star (1996) "How things (actor-net)work: Classification, magic and the ubiquity of standards", *Philosophia,* http://weber.ucsd.edu/~gbowker/actnet.html.

Bowker, G. and S. L. Star (1999) *Sorting Things Out: Classification and Its Consequences,* MIT press,

Bowman, I. T., R. C. Holt and N. V. Brewster (1999) "Linux as a Case Study: Its Extracted Software Architecture". in *The 21st International Conference on Software Engineering (ICSE 1999),*

Bradford, J. (2003 - Sun Mar 02) *Re: BitBucket: GPL-ed KitBeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0411.html.

Brey, P. (1997) "Philosophy of Technology Meets Social Constructivism", *Techne',* **2 (3-4).**

[illegible]

ffort in Online Groups: Who
terdisciplinary Perspectives,
ates, Mahwah, NJ.

ng the Dynamics of Science
lan, London.

omestication of the Scallops
Belief: A New Sociology of
don, pp. 196-233.

y as a Tool for Sociological
tems. New Directions in the
Hughes and T. J. Pinch eds)

pp. 387-399.

Butler, B., L. Sproull, S. Kiesler and R. Kraut (2002) "Community E
Does the Work and Why?" in *Leadership at a Distance: In*
(Weisband, S. and L. Atwater eds) Lawrence Erlbaum Associ

Callon, M. (1986a) "The Sociology of an Actor-Network" in *Mappi*
and Technology, (Callon, M., J. Law and A. Rip eds) Macmil

Callon, M. (1986b) "Some Elements of a Sociology of Translation: D
and Fishermen of St. Brieuc Bay" in *Power, Action and*
Knowledge?, (Law, J. ed.) (32) Routledge & Kegan Paul, Lon

Callon, M. (1987) "Society in the Making: The Study of Technolog
Analysis." in *The Social Construction of Technological Sys*
Sociology and History of Technology., (Bijker, W. E., T. P.
MIT press, Cambridge, MA.

Callon, M. (1999) "Actor-Network Theory - the market test" in *Actor Network Theory and After*, (Law, J. and J. Hassard eds) Blackwell Publishers / The Sociological Review, Oxford, pp. 181-195.

Callon, M. and B. Latour (1981) "Unscrewing the Big Leviathan: How Actors Macro-Structure Reality and How Sociologist Help Them To Do So." in *Advances in Social Theory and Methodology: Towards an Integration of Micro and Macro-Sociology*, (Knorr-Cetina, K. and A. V. Cicouvel eds) Routledge, Boston, MA; London, pp. 277-303.

Callon, M. and B. Latour (1992) "Don't Throw the Baby Out with the Bath School! A Reply to Collins and Yearley" in *Science as Practice and Culture*, (Pickering, A. ed.) Chicago University Press, Chicago.

Callon, M. and J. Law (1995) "Agency and the hybrid collectif", *South Atlantic Quarterly,* **94 (2),** pp. 481-507.

Campbell, D. and J. Stanley (1963) *Experimental and Quasi-Experimental Designs for Research.,* Houghton Mifflin, Boston, MA.

Cangelosi, V. and W. Dill (1965) "Organizational learning: Observations toward a theory", *Administrative Science Quarterly,* **10** p. 175–203.

Chia, R. (1998) "Introduction" in *In the Realm of Organization. Essays for Robert Cooper*, (Chia, R. ed.) Routledge, London, pp. 1-11.

Chia, R. (1999) "A 'Rhizomic' model of organizational change and transformation: Perspectives from a metaphysics of change", *British Journal of Management,* **10** pp. 209-227.

Chia, R. (2000) "Discourse Analysis as Organizational Analysis", *Organization,* **7 (3),** pp. 513-518.

Ciborra, C. (1999) "IT and Hospitality". in *IRIS 22 Proceedings Technical Report, TR-21,* University of Jyvaskyla,

Ciborra, C. U. (1996) "Improvisation and information technology in organizations". in *Proceedings of the seventeenth international conference on information systems,* Cleveland, Ohio, pp. 369-380, ACM.

Ciborra, C. U. (Ed.) (2000) *From Control to Drift,* Oxford University Press, Oxford.

Clegg, S., M. Kornberger and C. Rhodes (2005) "Learning/Becoming/Organizing", *Organization,* **12 (2),** pp. 147-167.

Clemm, G. (1989) "Replacing Version-Control with Job-Control". in *Proceedings of the 2nd International Workshop on Software configuration management,* Princeton, New Jersey, United States, pp. 162-169,

Collins, B. (2003 - Tue Mar 11) *Re: [ANNOUNCE] BK->CVS (real time mirror),* University of Indiana http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/0894.html.

Collins, H. M. and S. Yearley (1992a) "Epistemological Chicken" in *Science as Practice and Culture*, (Pickering, A. ed.) University of Chicago Press, Chicago, pp. 301-326.

Collins, H. M. and S. Yearley (1992b) "Journey into Space" in *Science as Practice and Culture*, (Pickering, A. ed.) University of Chicago Press, Chicago, pp. 369-389.

Constant, E. W. (1999) "Reliable Knowledge and Unreliable Stuff: On the Practical Role of Rational Beliefs", *Technology and Culture,* **40 (April),** pp. 324-357.

Cooper, R. (1976) "The Open Field", *Human Relations,* **29 (11),** pp. 999-1017.

Corbet (2002) *BitKeeper free use license,* Eklektix, Inc. http://lwn.net/Articles/12121/.

Cordella, A. and M. Shaikh (2003) "Actor Network Theory and After: What's New For IS Research?" in *European Conference of Information Systems,* Naples, Italy,June, 2003, Cox, A. (2000 - Wed Sep 13) *Re: Proposal: Linux Kernel Patch Management System,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0009.1/1076.html.

Coyne, I. T. (1997) "Sampling in qualitative research. Purposeful and theoretical sampling; merging or clear boundaries?" *Journal of Advanced Nursing,* **26** p. 623–630.

Crotty, M. (1998) *The Foundations of Social Research: Meaning and Perspective in the Research Process,* Sage Publications Ltd,

Crowston, K., H. Annabi, J. Howison and C. Masango (2005) "Effective work practices for FLOSS development: A model and propositions". in *38th Hawaii International Conference on System Sciences,* Hawaii,

Crowston, K. and J. Howison (2004) *The social structure of Free and Open Source software development.,* MIT Working Paper http://opensource.mit.edu/papers/crowstonhowison.pdf.

Crowston, K. and J. Howison (2006) "Hierarchy and Centralization in Free and Open Source Software Team Communications", *Knowledge, Technology, and Policy,* **18 (4),** pp. 65-85.

Crowston, K. and B. Scozzi (2002) "Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development", *IEE Proceedings - Software,* **149 (1),** pp. 3-17.

Cyert, R. M. and J. G. March (1963) *A Behavioral Theory of the Firm,* Prentice-Hall, Englewood Cliffs NJ.

Czarniawska, B. (1998) *A Narrative Approach to Organization Studies,* Sage, Thousand Oaks, CA.

Dahlander, L. (2004) *Appropriating the commons: firms in Open Source Software,* MIT Working Paper http://opensource.mit.edu/papers/dahlander2.pdf.

Dalle, M. and P. A. David (2005) "The Allocation of Software Development Resources in 'Open Source' Production Mode" in *Perspectives on Open Source and Free Software,* (Feller, J., B. Fitzgerald, S. Hissam and K. Lakhani eds) O'Reilly Associates, Sebastapol, CA.

David, K. and B. Pfaff (1998) *Society and Open Source: why Open Source software is better for society than closed source software,* http://www.msu.edu/user/pfaffben/writings/anp/oss-is-better.html.

Davidson, E. and M. Chiasson (2005) "Contextual influences on technology use mediation: a comparative analysis of electronic medical record systems", *European Journal of Information Systems,* **14** p. 6–18.

Day, G. S. (1994) "Continuous Learning about Markets", *California Management Review,* **36**.

de Certeau, M. (1988) *The Practice of Everyday Life,* University of California Press, Berkeley, California.

Deci, E. L. (1975) *Intrinsic Motivation,* Plenum Press, New York.

DeFillippi, R. and S. Ornstein (2003) "Psychological Perspectives Underlying Theories of Organizational Learning" in *Handbook of Organizational Learning and Knowledge,* (Easterby-Smith, M. and M. Lyles eds) Blackwell Press, Oxford, England.

Doolin, B. (1998) "Information Technology as Disciplinary Technology: Being Critical in Interpretive Research on Information Systems", *Journal of Information Technology,* **13 (4),** pp. 301-311.

Doolin, B. (2003) "Narratives of Change: Discourse, Technology and Organization", *Organization,* **10 (4),** pp. 751-770.

Doolin, B. and A. Lowe (2002) "To reveal is to critique: actor-network theory and critical information systems research", *Journal of Information Technology,* **17 (2),** pp. 69-78.

Dunford, R. and D. Jones (2000) "Narratives in Strategic Change", *Human Relations,* **53 (9),** pp. 1207-1226.

Dutta, J. and K. Prasad (2004) *Learning, ownership, and innovation in Open Source communities,* University of Florida Working Paper http://garnet.acns.fsu.edu/~kprasad/oss.pdf.

Easterby-Smith, M. and M. A. Lyles (2003) *The Blackwell Handbook of Organizational Learning and Knowledge Management,* Blackwell, Oxford.

Edwards, K. (2001) *Epistemic communities, situated learning and Open Source software development,* MIT Working Paper http://opensource.mit.edu/papers/kasperedwards-ec.pdf.

Elam, M. (1994) "Anti Anticonstructivism or Laying the Fears of a Langdon Winner to Rest", *Science, Technology, & Human Values,* **19 (1),** pp. 101-106.

Feldman, M. (2000) "Organizational routines as a source of continuous change." *Organization Science,* **11 (6),** p. 611–629.

Feller, J. and B. Fitzgerald (2002) *Understanding Open Source Software Development,* Addison-Wesley, London, UK.

Fielding, R., I. Hann, J. Roberts and S. Slaughter (2002) "Why do developers contribute to open source projects? First evidence of economic incentives". in *2nd Workshop on Open Source Software Engineering: Meeting Challenges and Surviving Success,* Orlando, FL, US,May 19-25,

Fielding, R. T. (1999) "Shared Leadership in the Apache Project", *Comunications of the ACM,* **42 (4).**

Fink, M. (2003) *The business and economics of Linux and Open Source,* Prentice Hall, Upper Saddle River, NJ.

Fitzgerald, B. (2006) "The Transformation of Open Source Software", *MIS Quarterly,* **30 (3),** pp. 587-598.

Fitzgerald, B. and P. J. Agerfalk (2005) "The Mysteries of Open Source Software: Black and White and Red All Over?" in *HICSS, Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05),* Hawaii, pp. 196a, Track 7,

Fitzgerald, B. and J. Feller (2001) "Guest editorial - Open source software: investigating the software engineering, psychosocial and economic issues", *Information Systems Journal,* **11 (4),** pp. 273-276.

Fitzgerald, B. and D. Howcroft (1998) "Towards Dissolution of the IS Research Debate: From Polarization to Polarity", *Journal of Information Technology,* **13** pp. 313-326.

Fleck, L. (1979) *Genesis and Development of a Scientific Fact,* The University of Chicago Press, Chicago.

Fogel, K. (1999) *Open Source Development with CVS,* Coriolis Open Press, Scottsdale, AZ.

Fogel, K. (2005) *Producing Open Source Software: How to Run a Successful Free Software Project,* O'Reilly, Sebastopol, CA.

Foucault, M. (1995) *Discipline and punish: The birth of the prison,* Vintage Books, New York.

Franck, E. and C. Jungwirth (2002) *Reconciling investors and donators - The governance structure of open source,* Working Paper Series, Chair of Strategic Management and Business Policy, University of Zurich. http://www.unizh.ch/ifbf/webFuehrung/Dokumente/WorkingPaper/FranckJungwirthInvestorsAndDonatorsWP8.pdf.

Franke, N. and E. Von Hippel (2003) "Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software", *Research Policy,* **32 (7),** pp. 1199-1215.

Gadamer, H. G. (1975) *Truth and Method,* The Seabury Press, New York.

Galbraith, J. R. (1977) *Organisation Design,* Addison-Wesley,

Gall, T. (2002- Fri Oct 04) *New BK License Problem?,* University of Indiana http://www.uwsg.iu.edu/hypermail/linux/kernel/0210.0/1496.html.

Gallivan, M. J. (2001) "Striking a Balance Between Trust and Control in a Virtual Organization: A Content Analysis of Open Source Software Case Studies", *Information Systems Journal,* **11** pp. 277-304.

Garcia, J. M. (2001) *INNOVATING WITHOUT MONEY:Linux and the Open Source Paradigm as an Alternative to Commercial Software Development,* SPRU - Science and Technology Policy Research University of Sussex http://opensource.mit.edu/papers/mateos.pdf.

Garcia, L. and F. Quek (1997) "Qualitative Research in Information Systems: Time to be Subjective?" in *Information Systems and Qualitative Research*, (Liebenau, J., A. Lee and J. DeGross eds) Chapman and Hall, London, pp. 444-465.

Garzik, J. (2003 - Sat Mar 01) *Re: BitBucket: GPL-ed KitBeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0147.html.

Geertz, C. (1973) *The Interpretation of Cultures,* Basic Books, New York.

Ghosh, R. A. (1998) "Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet", *FirstMonday: Peer Reviewed Journal on the Internet,* **3 (3),** pp. 1-19.

Gill, R. (2000) "Discourse Analysis" in *Qualitative Researching with Text, Image and Sound: A Practical Handbook*, (Bauer, M. W. and G. Gaskell eds) Sage, London, pp. 172-190.

Glaser, B. G. and A. Strauss (1967) *The Discovery of Grounded Theory: Strategies for Qualitative Research,* Aldine, Chicago.

Grand, S., G. Von Krogh, D. Leonard and W. Swap (2004) "Resource Allocation Beyond Firm Boundaries: A Multi-Level Model for Open Source Innovation", *Long Range Planning,* **37** pp. 591-610.

Grant, D., T. Keenoy and C. Oswick (1998) "Organizational Discourse: Of Diversity, Dichotomy and Multi-Disciplinarity" in *Discourse and Organization*, (Grant, D., T. Keenoy and C. Oswick eds) Sage, London, pp. 1-13.

Green, L. G. (1999) *Economics of Open Source Software,* http://badtux.org/home/eric/editorial/economics.php.

Hardy, C. (2004) "Scaling Up and Bearing Down in Discourse Analysis: Questions Regarding Textual Agencies and their Context", *Organization,* **11 (3),** pp. 415-425.

Hargadon, A. and A. Fanelli (2002) "Action and Possibility: Reconciling Dual Perspectives of Knowledge in Organizations", *Organization Science,* **13 (3),** pp. 290-302.

Hars, A. and S. Ou (2001) "Working for Free? - Motivations of Participating in Open Source Projects". in *34th Hawaii International Conference on Systems Sciences,* Hawaii, IEEE.

Hars, A. and S. Ou (2002) "Working for Free? Motivations for Participating in Open-Source Projects", *International Journal of Electronic Commerce,* **6 (3),** p. 25–39.

Hasselbladh, H. and F. Theodoridis (1998) "Social Magmas and the Conventional Explanatory Pyramid", *Scandinavian Journal of Management,* **14 (1-2),** pp. 53-76.

Hawkins, R. (2002) *The economics of the Open Source Software for a competitive firm,* MIT Working Paper http://opensource.mit.edu/papers/hawkins.pdf.

Hedberg, B. (1981) "How organizations learn and unlearn" in *Handbook of organizational behaviour*, (Nystrom, P. C. and W. H. Starbuck eds) Oxford University Press, New York, pp. 3-27.

Hedberg, B., P. C. Nystrom and W. H. Starbuck (1976) "Camping on See-saws: Prescriptions for a Self-designing Organization", *Administrative Science Quarterly,* **21 (1),** p. 41–65.

Hellwig, C. (2003 - Sat Mar 01) *Re: BitBucket: GPL-ed KitBeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0092.html.

Hemetsberger, A. and C. Reinhardt (2004) "Sharing and Creating Knowledge in Open-Source Communities The case of KDE". in *The Fifth European Conference on Organizational Knowledge, Learning, and Capabilities,* Innsbruck, Austria, MIT Open Source Site.

Henkel, J. (2004) *Patterns of free revealing - balancing code sharing and protection in commercial Open Source development,* MIT Working Paper http://opensource.mit.edu/papers/henkel2.pdf.

Henson, V. and J. Garzik (2002) "BitKeeper for Kernel Developers". in *Ottawa Linux Symposium,* Ottawa, Ontario Canada,June 26th–29th, 2002, pp. 197-212,

Heracleous, L. and M. Barrett (2001) "Organizational Change as Discourse: Communicative Actions and Deep Structures in the Context of Information Technology Implementation", *Academy of Management Journal,* **44 (4),** pp. 755-778.

Herbsleb, J. D. and R. E. Grinter (1999) "Splitting the organization and integrating the code: Conway's law revisited." in *21st International Conference on Software Engineering (ICSE 99),* Los Angeles,

Hertel, G., S. Niedner and S. Herrmann (2003) "Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel", *Research Policy: Special Issue on Open Source Software Development,* **32 (7),** pp. 1159-1178.

Himanen, P. (2001) *The Hacker Ethic and the Spirit of the Information Age,* Secker & Warburg, London, UK.

Holzner, B. and J. Marx (1979) *Knowledge application: the knowledge system in society,* Allyn and Bacon, Boston.

Huber, G. P. (1991) "Organizational Learning: The Contributing Processes and the Literatures", *Organization Science,* **2 (1),** pp. 88-115.

Hume, D. (1978) *A Treatise of Human Nature,* Clarendon, Oxford.

Iannacci, F. (2003) *The Linux Managing Model,* MIT Open Source Repository http://opensource.mit.edu/papers/iannacci2.pdf.

Jacobson, P. (2003) "It's all in the Mind". in *12th National Conference on Vocational and Educational Training Research,* Perth, Western Australia,10 July 2003,

Johnson, J. P. (2001) *Economics of Open Source Software,* http://opensource.mit.edu/papers/johnsonopensource.pdf.

Jones, M. (1995) "Organisational Learning: Collective Mind or Cognitivist Metaphor", *Accounting, Management and Information Technology,* **5 (1),** pp. 61-77.

Jones, M. (1998) "Information Systems and the Double Mangle: Steering a Course Between the Scylla of Embedded Structure and the Charybdis of Strong Symmetry". in *IFIP WG8.2 & WG8.6 Joint Working Conference on Information Systems: Current Issues and Future Changes,* Helsinki, Finland,December 10-13, 1998, p. 287, IFIP.

Jorgensen, N. (2001) "Putting it all in the Trunk: Incremental Software Development in the FreeBSD Open Source Project", *Information Systems Journal,* **11** pp. 321-336.

Kavanagh, D. and L. Araujo (1995) "Chronigami: Folding And Unfolding Time." *Accounting, Management and Information Technology,* **5 (2),** pp. 103-121.

Keats, D. (2003) "Collaborative development of open content: A process model to unlock the potential for African universities", *First Monday,* **8 (2).**

Kendall, J. (1999) "Axial Coding and the Grounded Theory Controversy", *Western Journal of Nursing Research,* **21 (6),** pp. 743-757.

Kesan, J. P. and R. C. Shah (2002) *SHAPING CODE,* MIT http://opensource.mit.edu/papers/shah.pdf.

Kiesler, S. and L. Sproull (1982) "Managerial Responses to Changing Environments: Perspectives on Problem Sensing from Social Cognition", *Administrative Science Quarterly,* **27 (4),** pp. 548-570.

Kilpi, T. (1997) "New Challenges for Version Control and Configuration Management: a Framework and Evaluation", *IEEE Computer,* **(1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97)),** pp. 33-41.

Kiskis, M. and R. Petrauskas (2005) "Lessig's implications for intellectual property law and beyond them", *International Review of Law, Computers & Technology,* **19 (3),** pp. 305-316.

Klein, H. and D. P. Truex (1996) "Discourse Analysis: An Approach to the Investigation of Organizational Emergence" in *Signs of Work Semiosis and Information Processing in Organizations*, (Holmqvist, B., H. Klein and R. Posner eds) Walter de Gruyter, Berlin.

Klein, H. K. and M. D. Myers (1999) "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems", *MIS Quarterly,* **(March),** pp. 67-93.

Kleve, P. and R. Mulder (2005) "Code is Murphy's law", *International Review of Law, Computers & Technology,* **19 (3),** pp. 317-328.

Knorr Cetina, K. (1993) "Strong Constructivism - From a Sociologist's Point of View", 'A Personal Addendum to Sismondo's Paper'", *Social Studies of Science,* **23.**

Koch, S. and G. Schneider (2002) "Effort, Cooperation and Coordination in an Open Source Software Project: GNOME", *Information Systems Journal,* **12 (1),** pp. 27-42.

Kogut, B. and A. Metiu (2001) "Open-source software development and distributed innovation", *Oxford Review of Economic Policy,* **17 (2),** pp. 248-264.

Koike, H. and H. C. Chu (1997a) "VRCS: Integrating Version Control and Module Management using Interactive Three-Dimensional Graphics". in *Proceedings of the IEEE Symposium on Visual Languages (VA '97),* pp. 168-173, IEEE.

Koike, H. and H.-C. Chu (1997b) "Integrating Version Control and Module Management Using Three-Dimensional Visualization." in *HCI,* San Francisco, California, USA, pp. 853-856,

Kollock, P. (1999a) "The economics of on line cooperation: gift and public goods in cyberspace" in *Communities in cyberspace*, (Smith, Marc and Kollock eds) Routledge, London, pp. 220-246.

Kollock, P. (1999b) "The Production of Trust in Online Markets" in *Advances in Group Processes*, (Lawler, E. J., M. Macy, S. Thyne and H. A. Walker. eds) JAI Press, Greenwich, CT.

Kornberger, M. and S. Clegg (2004) "Bringing space back in: organizing the generative building", *Organization Studies,* **25** pp. 1095-1114.

Krippendorff, K. (1980) *Content Analysis: An Introduction to its Methodology,* Sage Publications, Inc, Beverly Hills, California.

Kuster, B., M. Osterloh and S. Rota (2002) *Trust and commerce in Open Source - a contradiction?,* MIT Working Paper http://opensource.mit.edu/papers/osterlohrotakuster.pdf.

Lakhani, K. and E. von Hippel (2003) "How Open Source software works: "Free" user-to-user assistance", *Research Policy,* **32** pp. 923-943.

Lakhani, K., B. Wolf, J. Bates and C. DiBona (2003) "The Boston Consulting Group hacker survey". in *Linux Conference Australia 2003,* Perth, Australia,January 22-25, 2003,

Lakhani, K. R. and R. G. Wolf (2005) "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects" in *Perspectives on Free and Open Source Software,* (Feller, J., B. Fitzgerald, S. Hissam and K. R. Lakhani eds) MIT Press.

Latour, B. (1986) "The Powers of Association" in *Power, Action and Belief. A New Sociology of Knowledge? Sociological Review Monograph 32*, (Law, J. ed.) Routledge & Kegan and Paul, London, pp. 264-280.

Latour, B. (1987) *Science in Action: How to follow Scientists and Engineers Through Society,* Bantam, New York.

Latour, B. (1988a) "Mixing humans and nonhumans together: the sociology of a door-closer", *Social Problems,* **35 (3),** pp. 298-310.

Latour, B. (1988b) "The Prince for Machines as well as for Machinations" in *Technology and Social Process*, (Elliot, B. ed.) Edinburgh University Press, Edinburgh, pp. 20-43.

Latour, B. (1988c) "A Relativistic Account of Einstein's Relativity", *Social Studies of Science,* **18 (1),** pp. 3-44.

Latour, B. (1991) "Technology is Society Made Durable" in *A Sociology of Monsters: Essays on Power, Technology and Domination*, (Law, J. ed.) Routledge, London.

Latour, B. (1992) "Pasteur on Lactic Acid Yeast: A Partial Semiotic Analysis", *Configurations,* **1 (1),** pp. 129-145.

Latour, B. (1993) *We Have Never Been Modern,* Harvester Wheatsheaf, Hemel Hempstead.

Latour, B. (1994) "Where are the missing masses? The sociology of a few mundane artifacts" in *Shaping technology / building society: studies in sociotechnical change*, (Bijker, W. E. and J. Law eds) The MIT Press, Cambridge Ma, pp. 225-258.

Latour, B. (1995) "Social theory and the study of computerized work sites" in *Information technology and changes in organizational work*, (Orlikowski, W. J., G. Walsham, M. R. Jones and J. I. DeGross eds) Chapman & Hall, London, pp. 295-307.

Latour, B. (1996) "Social Theory and the Study of Computerized Work Sites" in *Information Technology and Changes in Organizational Work*, (Orlikowski, W. J., G. Walsham, M. R. Jones and J. I. DeGross eds) Chapman and Hall, London.

Latour, B. (1999a) "Give Me a Laboratory and I Will Raise the World" in *The Science Studies Reader*, (Biagioli, M. ed.) Routledge, New York/London, pp. 258-275.

Latour, B. (1999b) "On Recalling ANT" in *Actor Network and After*, (Law, J. and J. Hassard eds) Blackwell and the Sociological Review, Oxford.

Latour, B. (1999c) *Pandora's Hope: Essays on the Reality of Science Studies,* Harvard University Press, Cambridge, MA.

Latour, B. (2001) *ANT Lectures at the LSE* (Personal communication).

Latour, B. (2002) "Morality and Technology: The End of the Means", *Theory, Culture and Society,* **19 (5/6),** pp. 247-260.

Latour, B. (2003) "The Promises of Constructivism" in *Chasing Technoscience: Matrix for Materiality*, (Ihde, D. and E. Selinger eds) Indiana University Press, Bloomington, IN.

Latour, B. (2004a) "On Using ANT for Studying Information Systems: A (Somewhat) Socratic Dialogue" in *The Social Study of Information and Communication Technology: Innovation, Actors, and Contexts*, (Avgerou, C., C. U. Ciborra and F. F. Land eds) Oxford University Press, Norfolk, UK, pp. 62-76.

Latour, B. (2004b) "Why Has Critique Run out of Steam? From Matters of Fact to Matters of Concern", *Convergencia,* **11 (35),** pp. 17-49.

Latour, B. (2005a) "From Realpolitik to Dingpolitik - or How to Make Things Public" in *Making Things Public- Atmospheres of Democracy*, (Latour, B. and P. Weibel eds) MIT Press.

Latour, B. (2005b) *Reassembling the Social: An Introduction to Actor-Network-Theory (Clarendon Lectures in Management Studies),* Oxford University Press, Oxford.

Latour, B. (2005c) *What is the style of matters of concern? Two lectures in empirical philosophy,* Two lectures given in Amsterdam April and May 2005 for the Spinoza Chair in Philosophy, University of Amsterdam http://www.ensmp.fr/~latour/articles/article/97-STYLE-MATTERS-CONCERN.pdf.

Latour, B. and J. Johnson (1995) "Mixing humans and nonhumans together: The sociology of door-closer" in *Ecologies of knowledge: work and politics in science and technology*, (Star, S. L. ed.) SUNY Press., pp. 257-277.

Latour, B. and P. Weibel (Eds.) (2005) *Making Things Public: Atmospheres of Democracy,* MIT Press, Cambridge, MA.

Lave, J. and E. Wenger (1991) *Situated learning : legitimate peripheral participation,* Cambridge University Press, Cambridge England ; New York.

Law, J. (1991) "Introduction: Monsters, Machines and Sociotechnical Relations" in *A Sociology of Monsters: Essays on Power, Technology and Domination*, (Law, J. ed.) Routledge, London.

Law, J. (1992) "Notes on the Theory of the Actor-Network: Ordering, Strategy and Heterogeneity", *Systems Practice,* **5 (4),** pp. 379-393.

Law, J. (1994) *Organizing Modernity,* Basil Blackwell, Oxford.

Law, J. (1999) "After ANT: Topology, Naming and Complexity" in *Actor Network Theory and After*, (Law, J. and J. Hassard eds) Blackwell and the Sociological Review, Oxford and Keele.

Law, J. (2002) *Aircraft Stories: Decentering the Object in Technoscience,* Duke University Press, Law, J. (2004) *After Method: Mess in Social Science Research,* Routledge, Oxford.

Law, J. and W. Bijker (1994) "Postscript: technology, stability, and social theory" in *Shaping technology / building society: studies in sociotechnical change*, (Bijker, W. E. and J. Law eds) The MIT Press, Cambridge Ma, pp. 290-308.

Law, J. and M. Callon (1988) "Engineering and Sociology in a Military Aircraft Project: A Network Analysis of Technological Change", *Social Problems,* **35 (3),** pp. 284-297.

Law, J. and V. Singleton (2000) "Performing Technology's Stories: On Social Constructivism, Performance, and Performativity", *Technology and Culture,* **41 (4),** pp. 765-775.

Lee, A. S. (1991) "Integrating Positivist and Interpretive Approaches to Organizational Research", *Organization Science,* **4** pp. 342-365.

Lee, A. S. and R. Baskerville (2003) "Generalizing Generalizability in Information Systems Research", *Information System Research,* **14 (3),** pp. 221-243.

Lee, A. S., J. Liebenau and J. I. DeGross (1997) "Preface: Information Systems and Qualitative Research" in *Information Systems and Qualitative Research*, (Lee, A. S., J. Liebenau and J. I. DeGross eds) Chapman and Hall, London.

Lee, H. (1999) "Time and Information Technology: Monochronicity, Polychronicity and Temporal Symmetry", *European Journal of Information Systems,* **8** pp. 16-26.

Lee, H. and J. Liebenau (2000) "Time and the Internet at the Turn of the Millennium", *Time and Society,* **9 (1),** pp. 43-56.

Lee, N. and J. Hassard (1999) "Organization unbound: actor-network theory, research strategy and institutional flexibility", *Organization,* **6 (3),** pp. 391-404.

Lee, S., N. Moisa and M. Weiss (2003) *Open Source as a signalling device - an economic analysis,* MIT Working Paper http://opensource.mit.edu/papers/leemoisaweiss.pdf.

Leenes, R. and B.-J. Koops (2005) "'Code': Privacy's death or saviour?" *International Review of Law, Computers & Technology,* **19 (3),** pp. 329-340.

Lerner, J. (2002) *Incentives and Open Source software,* http://cip.umd.edu/lerner.htm.

Lerner, J. and J. Tirole (2000) "The Simple Economics of Open Source" *National Bureau of Economic Research* Working Paper 7600

Lerner, J. and J. Tirole (2001) "The Open Source movement: key research questions", *European Economic Review,* **45** pp. 819-826.

Lerner, J. and J. Tirole (2002a) *The Scope of Open Source Licensing,* MIT Working Paper http://opensource.mit.edu/papers/lernertirole2.pdf.

Lerner, J. and J. Tirole (2002b) "Some simple economics of the Open Source", *The Journal of Industrial Economics,* **2** pp. 197-234.

Lerner, J. and J. Tirole (2004) "The economics of technology sharing: Open source and Beyond," *Harvard Business School* 04-35

Lessig, L. (1999a) *Code and Other Laws of Cyberspace,* Basic Books,

Lessig, L. (1999b) "Code and the Commons". in *Conference of Media Convergence,* Fordham Law School, Fordham University, New York, NY,Feb 9, 1999,

Lessig, L. (1999c) "The Limits in Open Code: Regulatory Standards and the Future of the Net", *Berkeley Technology Law Journal,* **14 (2).**

Lessig, L. (2001) *The Future of Ideas: The Fate of the Commons in a Connected World,* Random House,

Lincoln, Y. S. and E. G. Guba (1985) *Naturalistic Inquiry,* Sage Publications, Beverly Hills.

Lindblom, C. E. (1959) "The Science of 'Muddling Through'", *Public Administration Review,* **19** pp. 79-88.

Lipset, D. (1980) *Gregory Bateson: The legacy of a scientist,* Prentice Hall, Englewood Cliffs, NJ.

Ljungberg, J. (2000) "Open Source Movements as a Model for Organizing", *European Journal of Information Systems,* **9 (4),** pp. 208-216.

Louridas, P. (2006) "Version Control", *IEEE Software,* **January/February** pp. 104-107.

Machek, P. (2003 - Mon Mar 03) *Re: BitBucket: GPL-ed KitBeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0422.html.

Machek, P. (2003 - Wed Feb 26) *BitBucket: GPL-ed BitKeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0302.3/0931.html.

MacKenzie, D. (1990) *Inventing Accuracy: A Historical Sociology of Nuclear Missile Guidance,* MIT Press, Cambridge, Mass.

MacKenzie, D., P. Eggert and R. Stallman (2002) *Comparing and Merging Files with GNU diff and patch: for Diffutils 2.8.1 and patch 2.5.4, 5th April 2002,* Network Theory Ltd, Bristol, UK.

MacKenzie, D. and J. Wajcman (1985) "Introduction: The Social Shaping of Technology" in *The Social Shaping of Technology.*, (MacKenzie, D. and J. Wajcman eds) Open University Press, Milton Keynes.

Markus, M. L., B. Manville and C. E. Agres (2000) "What Makes a Virtual Organization Work?" *Sloan Management Review,* **42 (1),** pp. 13-26.

Martin de Holan, P. and N. Phillips (2003) "Organizational Forgetting" in *Handbook of Organizational Learning*, (Easterby-Smith, M. and M. Lyles eds) Sage, New York.

Maslow, A. H. (1959) *Motivation and Personality,* New York,

Massey, B. (2003) *Open Source workshop, Portland, Oregon* (Personal communication).

Mauss, M. (1954) *The Gift: Forms and Functions of Exchange in Archaic Societies,* Cohen and West Ltd, London.

McVoy, L. (1999 - Sun, 21 Feb) *revision control for the kernel (BitKeeper),* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/9902.2/0812.html.

McVoy, L. (2002 - Fri Oct 04) *Re: New BK License Problem?,* University of Indiana http://www.uwsg.iu.edu/hypermail/linux/kernel/0210.0/1500.html.

McVoy, L. (2003 - Sat Mar 01) *Re: BitBucket: GPL-ed BitKeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0052.html.

Miller, D. S. (1997 - Sun, 14 Sep) *First vger CVS 2.1.x kernel source repository snapshot,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/9709.1/0432.html.

Mockus, A., R. T. Fielding and J. Herbsleb (2002) "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology (TOSEM),* **11 (3),** pp. 309 - 346.

Moglen, E. (2001) "Enforcing the GPL, II". in *LinuxUser,* 15), October.

Mol, A. (1999) "Ontological politics. A word and some questions" in *Actor Network Theory and After*, (Law, J. and J. Hassard eds) Blackwell Publishers / The Sociological Review, Oxford, pp. 74-89.

Mol, A. (2002) *The Body Multiple: Ontology in Medical Practice,* Duke University Press, Durham, NC.

Mol, A. and J. Law (1994) "Regions, Networks and Fluids: Amaemia and Social Topology", *Social Studies of Science,* **24** pp. 641-671.

Molnar, I. (2002 - Sun Oct 06) *BK MetaData License Problem?,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0210.0/1918.html.

Monteiro, E. and O. Hanseth (1995a) "Social shaping of information infrastructure: On being specific about the technology" in *Information technology and changes in organizational work*, (Orlikowski, W. J., G. Walsham, M. R. Jones and J. I. DeGross eds) Chapman & Hall, London, pp. 325-343.

Monteiro, E. and O. Hanseth (1995b) "Standardisation in action: Achieving universalism and localisation in medical protocols" in *Standardisation in action: Achieving universalism and localisation in medical protocols*, (Orlikowski, W. J., J. Walsh and J. I. De Gross eds) Chapman a& Hall, London, pp. 325-343.

Moody, G. (2001) *Rebel Code: Linux and the Open Source Revolution,* Penguin, London.

Moon, J. Y. and L. Sproull (2000) "Essence of Distributed Work: The Case of the Linux Kernel", *First Monday,* **5 (11).**

Moser, I. and J. Law (2006) "Fluids or flows? Information and qualculation in medical practice", *Information Technology & People,* **19 (1),** pp. 55-73.

ory of economic change, Belknap Press

Sage, California.

Building a foundation for Knowledge

Nelson, R. R. and S. G. Winter (1982) *An evolutionary thec* of Harvard University Press, Cambridge, Mass.

Neuendorf, K. A. (2002) *The Content Analysis Guidebook,*

Nonaka, I. and N. Konno (1998) "The Concept of "Ba":

Orlikowski, W. J. and J. D. Hofman (1997) "An improvisational model for change management: The case of groupware technologies", *Sloan Management Review,* **38 (2),** pp. 11-21.

Ormrod, S. (1995) "Feminist sociology and methodology:leaky black boxes in gender/technology relations" in *The Gender–Technology Relation: Contemporary Theory and Research*, (Grint, K. and R. Gill eds) Taylor & Francis, London, p. 31–47.

Orr, J. (1990a) "Sharing Knowledge, Celebrating Identity: War Stories and Community Memory in a Service Culture" in *Collective Remembering: Memory in Society*, (Middleton, D. S. and D. Edwards eds) Sage Publications, Beverley Hills, CA, pp. 169-189.

Orr, J. (1990b) Talking About Machines: An Ethnography of a Modern Job.

Patriotta, G. (2003) *Organizational Knowledge in the Making: How Firms Create, Use and Institutionalize Knowledge,* Oxford, New York : Oxford University Press.

Patriotta, G. (2004) "On studying organizational knowledge", *Knowledge Management Research & Practice,* **2 (1),** pp. 3-12.

Pavlicek, R. C. (2000) *Embracing Insanity: Open Source Software Development,* Sams, Indianapolis, IN.

Pels, D. (1995) "Have We Never Been Modern? Towards a Demontage of Latour's Modern Constitution", *History of the Human Sciences,* **8 (3),** pp. 129-141.

Penrose, E. T. (1959) *The theory of the growth of the firm,* Wiley, New York,.

Pentland, B. T. (1992) "Organising Moves in Software Support Hot Lines", *Administrative Science Quarterly,* **37** pp. 527-548.

Pentland, B. T. (1995) "Information systems and organizational learning: The social epistemology of organizational knowledge systems", *Accounting, Management and Information Technologies,* **5 (1),** pp. 1-21.

Perens, B. (1997) *The Debian Free Software Guidelines,* Open Source Initiative http://www.opensource.org/docs/definition.php.

Perens, B. (1999) "The Open Source Definition" in *Open Sources: Voices from the Open Source Revolution*, (DiBona, C., S. Ockman and M. Stone eds) O'Reilly, CA.

Peters, T. (1994) "To forget is sublime", *Forbes,* **April 11** pp. 128-130.

Phillips, N. and C. Hardy (2002) *Understanding Discourse Analysis: Investigating Processes of Social Construction,* Sage, Thousand Oaks, CA.

Pickering, A. (1993) "The Mangle of Practice: Agency and Emergence in the Sociology of

ngle of Practice: Time, Agency and Science,* University of Chicago

(1987) "The Social Construction of Facts and Artifacts: Or How the and the Sociology of Technology Might Benefit Each Other" in *The f Technological Systems*, (Bijker, W. E., T. Pinch and T. P. Hughes , pp. 17-50.

Analysis as a Way of Analysing Naturally Occurring Talk" in (Silverman, D. ed.) Sage, London, pp. 144-160.

ing up to Open Source", *The Richmond Journal of Law and* 24-.

ley (2000) "Representing human and non-human stakeholders: On rity". in *Organizational and social perspectives on information* Denmark,June 10-12, 2000, pp. 339-354, Kluwer.

Pickering, A. (1995) *The Ma*
Press, Chicago.

Pinch, T. J. and W. E. Bijker
Sociology of Science
Social Construction o
eds) Cambridge, Mas

Potter, J. (1997) "Discourse
Qualitative Research,

Potter, S. W. (2000) "Oper
Technology, **6 (5),** pp.

Pouloudi, A. and E. A. Whit
speaking with autho
technology, Aalborg,

Pries-Heje, J., R. Baskerville, L. Levine and B. Ramesh (2004) "The High Speed Balancing Game: How Software Companies Cope with Internet Speed", *Scandinavian Journal of Information Systems,* **16** pp. 11-54.

Quah, D. (2002a) "Matching demand and supply in a weightless economy: Market-based creativity with and without IPRs" *Centre for Economic Performance, LSE* London.

Quah, D. (2002b) "Technology Dissemination and Economic Growth: Some Lessons for the New Economy" *Centre for Economic Performance, LSE* London.

Quah, D. (2003) "Digital goods and the new economy" in *New Economy Handbook*, (Jones, D. ed.) Academic Press Elsevier Science, pp. 289-321.

Raymond, E. (1998a) *The Halloween Documents,* http://www.opensource.org/halloween.

Raymond, E. (1999a) *The Cathedral and the Bazaar: Musings on Linux and open source by an accidental revolutionary,* O'Reilly & Associates, Sebastopol, California.

Raymond, E. S. (1998b) "The Cathedral and the Bazaar", *FirstMonday: A Peer Reviewed Journal on the Internet,* **3 (3),** pp. 1-24.

Raymond, E. S. (1999b) *The Magic Cauldron,* http://www.tuxedo.org/~esr/writings/magic-cauldron/magic-cauldron.html.

Raymond, E. S. (2001) *The Cathedral & the Bazaar,* (2) O'Reilly, Sebastapol, CA.

Raymond, E. S. and W. C. Trader (1999) "Linux and Open-Source Success", *IEEE Software,* **16 (1),** pp. 85-89.

Reed, M. I. (1995) "The Action/Structure Debate in Organizational Analysis". in *Conference on Structuration Theory and Organizations,* Paris, France,

Robbins, J. (2005) "Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools" in *Perspectives on Free and Open Source Software*, (Feller, J., B. Fitzgerald, S. A. Hissam and K. R. Lakhani eds) MIT Press.

Roberts, J., I. Hann and S. Slaughter (2004) "Understanding the motivations, participation and performance of Open Source Software developers: a longitudinal study of the Apache projects" *Carnegie Mellon University Working Paper*

Robrecht, L. (1995) "Grounded Theory: Evolving Methods", *Qualitative Health Research,* **5 (2),** pp. 169-177.

Rochkind, M. J. (1975) "The Source Code Control System", *IEEE Transaction Software Engineering,* **1 (4),** pp. 364-370.

Rodríguez, O. M., *et al.* (2004) "Identifying Knowledge Management Needs in Software Maintenance Groups: A qualitative approach". in *Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04),*

Rose, J. and M. Jones (2004) "The Double Dance of Agency: a socio-theoretic account of how machines and humans interact". in *ALOIS 2004 - Action in Language, Organisations and Information Systems,* Linköping - Sweden,

Rosenberg, A. (1993) "Hume and the philosophy of science." in *The Cambridge Companion to Hume*, (Norton, D. ed.) Cambridge University Press, New York, pp. 64-89.

Rosenberg, D. K. (2000) *Open Source: The Unauthorized White Papers,* M & T Books, Foster City, CA.

Rossi, C. and A. Bonaccorsi (2005a) *Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach?,* MIT http://opensource.mit.edu/papers/rossi_motivations.pdf.

Rossi, C. and A. Bonaccorsi (2005b) "Intrinsic vs. extrinsic incentives in profit–oriented firms supplying Open Source products and services", *FirstMonday,* **10 (5)**.

Rothwell, W. J. (1983) "Bateson's Heterarchy of Learning", *Training and Development Journal,* **July** pp. 24-27.

Ryle, G. (1949) *The Concept of Mind,* Hutchinson, London, UK.

Sabel, C. F. (1982) *Work and politics: the division of labour in industry,* Cambridge University Press, Cambridge.

Sandelands, L. E. and R. E. Stablein (1987) "The Concept of Organization Mind" in *Research in the Sociology of Organizations,* (Bachrach, S. and N. DiTomaso eds), pp. 135-162.

Savirimuthu, J. (2005) "Open source, code and architecture: It is the Memes stupid", *International Review of Law, Computers & Technology,* **19 (3)**, pp. 341-362.

Schaffer, S. (1991) "The Eighteenth Brumaire of Bruno Latour", *Studies in History and Philosophy of Science,* **22** pp. 174-192.

Schon, D. (1975) "Deutero-Learning in Organizations: Learning for Increased Effectiveness", *Journal of Organizational Dynamics,* **4 (1)**, pp. 2-16.

Schön, D. A. (1983) *The Reflective Practitioner. How professionals think in action,* Temple Smith, London.

Senge, P. (1990) *The Fifth Discipline: The Art and Practice of the Learning Organization,* Doubleday, New York.

Sharma, S., V. Sugumaran and B. Rajagopalan (2002) "A framework for creating hybrid-open source software communities", *Information Systems Journal,* **12 (1)**, pp. 7-26.

Silva, L. and J. Backhouse (1997) "Becoming part of the furniture: The institutionalization of information systems". in *Information systems and qualitative research,* Philadelphia, pp. 389-414, Chapman and Hall.

Simon, H. A. (1991) "Bounded Rationality and Organizational Learning", *Organization Science,* **2** pp. 125-134.

Sims, H. P. and D. A. Gioia (1986) *The Thinking Organization,* Jossey-Bass., San Francisco.

Sismondo, S. (1993) "Some Social Constructions", *Social Studies of Science,* **23** pp. 515-553.

Sismondo, S. (1996) *Science Without Myth: On Constructions, Reality, and Social Knowledge,* State University of New York Press, New York.

Smith, M. K. (2001) *Chris Argyris: theories of action, double-loop learning and organizational learning,* INFED www.infed.org/thinkers/argyris.htm.

Spinellis, D. (2005) "Version Control Systems", *IEEE Software,* **September/October** pp. 108-109.

Stalder, F. (2000) "Beyond constructivism: towards a realistic realism. A review of Bruno Latour's Pandora's Hope", *The Information Society,* **16 (3)**, pp. 245-247.

Stallman, R. (1984) *The GNU Manifesto,* http://www.gnu.org/gnu/manifesto.html.

Stallman, R. (1999a) *The Free Software Definition,* Free Software Foundation http://www.gnu.org/philosophy/free-sw.html.

Stallman, R. (1999b) "The GNU Operating System and the Free Software Movement" in *Open Sources : Voices from the Open Source Revolution,* (DiBona, C., S. Ockman and M. Stone eds) O'Reilly.

Star, S. and K. Ruhleder (1996) "Steps toward an Ecology of Infrastructure: Design, Access for Large Information Space", *Information System Research,* **7 (1)**, pp. 111-134.

Star, S. L. (1991) "Power, Technologies and the Phenomenology of Conventions: On Being Allergic to Onions" in *A Sociology of Monsters: Essays on Power, Technology and Domination*, (Law, J. ed.) Routledge, London.

Stata, R. (1989) "Organizational Learning - The Key to Management Innovation", *Sloan Management Review,* **Spring** pp. 63-74.

Stewart, K. J. and S. Gosain (2006) "The Impact of Ideology on Effectiveness in Open Source Software Development Teams", *MIS Quarterly,* **30 (2),** p. 291–314.

Strauss, A. (1987) *Qualitative Analysis for Social Scientists,* Cambridge University Press, Cambridge.

Strauss, A. and J. Corbin (1999) *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory,* Sage Publications,

Suchman, L. (1987) *Plan and situated actions: the problem of human machine communication,* Cambridge University Press, Cambridge.

Suchman, L. (1998) "Human/Machine Reconsidered", *Cognitive Studies,* **5 (1),** pp. 5-13.

Swanson, E. B. (1976) "The Dimensions of Software Maintenance". in *Proceedings of the 2nd IEEE International Conference on Software Engineering,,* pp. 492-497,

Tenkasi, R. V. and R. J. Boland (1993) "Locating meaning making in organizational learning: The narrative basis of cognition", *Research in Organizational Change and Development,* **7** pp. 77-103.

Tichy, W. F. (1985) "RCS - a system for version control." *Software - Practice and Experience,* **15 (7),** pp. 637-654.

Tognetti, S. (1999) "Science in a double-bind: Gregory Bateson and the origins of post-normal science", *Futures,* **31** pp. 689-703.

Torvalds, L. (2002 - Sat Apr 20) *Re: [PATCH] Remove Bitkeeper documentation from Linux tree,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0204.2/1018.html.

Torvalds, L. (2003 - Fri Mar 07) *Re: BitBucket: GPL-ed KitBeeper clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/1932.html.

Torvalds, L. (2003 - Tue Mar 04) *Re: BitBucket: GPL-ed *notrademarkhere* clone,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0695.html.

Torvalds, L. and D. Diamond (2002) *Just for Fun,* Texere, London, UK.

Torvalds, L. and C. Schlenter (1995, 1996) *Re: CVS, Linus, and us,* University of Indiana http://www.uwsg.indiana.edu/hypermail/linux/kernel/9602/0800.html.

Tuomi, I. (2000) "Data is more than Knowledge: Implications of the Reversed Knowledge Hierarchy to Knowledge management and Organizational memory", *Journal of Management Information Systems,* **6 (3),** pp. 103-117.

Tuomi, I. (2002) *Networks of Innovation: Change and Meaning in the Age of the Internet,* Oxford University Press, Oxford.

Ungson, G., D. Braunstein and P. Hall (1981) "Managerial Information Processing: A Research Review", *Administrative Science Quarterly,* **26 (1),** pp. 116-134.

Valloppillil, V., J. Cohen and E. Raymond] (1998a) *Halloween Document I,* OpenSource.Org http://www.opensource.org/halloween/halloween1.html.

Valloppillil, V., J. Cohen and E. Raymond] (1998b) *Halloween Document II,* OpenSource.Org http://www.opensource.org/halloween/halloween2.html.

van der Hoek, A. (2000) "Configuration Management and Open Source Projects". in *Proceedings of the 3rd International Workshop on Software Engineering over the Internet,* Limerick, Ireland,

van Dijk, T. A. (1997a) "Discourse as Interaction in Society" in *Discourse as Social Interaction,* (van Dijk, T. A. ed.) Sage, London, pp. 1-37.

van Dijk, T. A. (1997b) "The Study of Discourse" in *Discourse as Structure and Process,* (van

Visser, M. (2003) "Gregory Bateson on Deutero-Learning and Double Bind: A Brief Conceptual History", *Journal of History of the Behavioral Sciences,* **39 (3),** p. 269–278.

von Hippel, E. (2002) "Open Source Projects as Horizontal Innovation Networks - By and For Users" *MIT Sloan Working Paper No. 4366-02*

von Hippel, E. (2005) *Democratizing Innovation,* MIT Press, Cambridge, MA.

von Hippel, E. and G. von Krogh (2003) "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science", *Organization Science,* **14 (2),** pp. 209-223.

von Krogh, G., S. Haefliger and S. Spaeth (2003a) *Collective Action and Communal Resources in Open Source Software Development: The Case of Freenet,* MIT Open Source http://opensource.mit.edu/papers/vonkroghhaefligerspaeth.pdf.

von Krogh, G., S. Spaeth and K. R. Lakhani (2003b) "Community, joining, and specialization in open source software innovation: a case study", *Research Policy,* **32 (7),** pp. 1217-1241.

Walsh, J. P. and G. R. Ungson (1991) "Organisational memory", *Academy of Management Review,* **16** pp. 57-91.

Walsham, G. (1993a) "Information Systems Strategy Formation and Implementation: The Case of a Central Government Agency", *Accounting, Management, & Information Technology,* **3 (3),** pp. 191-209.

Walsham, G. (1993b) *Interpreting Information Systems in Organizations,* John Wiley & Sons Ltd, Chichester.

Walsham, G. (1995a) "The emergence of interpretivism in IS research", *Information systems research,* **6 (4),** pp. 376-394.

Walsham, G. (1995b) "Interpretive Case Studies in IS Research: Nature and Method", *European*

Wayner, P. (2000) *Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans,* HarperCollins, New York, NY.

Weber, C. (2000) *The Political Economy of Open Source,* Berkley Roundtable on International Economy (BRIE) http://brie.berkeley.edu/pubs/wp/wp140.pdf.

Weber, S. (2004) *The Success of Open Source,* Harvard University Press,

Weber, S. (2005a) "Patterns of Governance in Open Source" in *Open Sources 2.0: The Continuing Evolution*, (DiBona, C., M. Stone and D. Cooper eds) O'Reilly, Sebastopol, CA, pp. 361-372.

Weber, S. (2005b) "The Political Economy of Open Source and Why it Matters" in *Digital Formations: IT and New Architectures in the Global Realm*, (Latham, R. and S. Sassen eds) Princeton University Press, Princeton, New Jersey, pp. 178-212.

Weick, K. (1995) *Sensemaking in Organizations,* Sage Publications, Thousand Oaks, CA.

Weick, K. E. (1979) *The social psychology of organizing,* (2nd) Addison-Wesley, Reading, MA.

Weick, K. E. (1988) "Enacted sensemaking in crisis situations", *Journal of Management Studies,* **25 (4),** pp. 305-317.

Weick, K. E. (1991) "The nontraditional quality of organizational learning", *Organization Science,* **2 (1),** pp. 116-124.

Weick, K. E. (1993) "The collapse of sensemaking in organizations: The Mann Gulch disaster", *Administrative Science Quarterly,* **38 (4),** pp. 628-652.

Weick, K. E. (1996) "Drop Your Tools: An Allegory for Organizational Studies", *Administrative Science Quarterly,* **41 (2),** pp. 301-313.

Weick, K. E. (1998) "Improvisation as a mindset for organizational analysis." *Organization Science,* **9** p. 543–555.

Weick, K. E. (2001) *Making Sense of the Organization,* Blackwell Publishers, MA.

Weick, K. E. and R. E. Quinn (1999) "Organizational change and development", *Annual Review of Psychology,* **50** p. 361–386.

Weick, K. E. and K. H. Roberts (1993) "Collective Mind in Organizations: Heedful Interrelating on Flight Decks", *Administrative Science Quarterly,* **38 (3),** pp. 357-381.

Weick, K. E. and F. Westley (1996) "Organizational Learning: Affirming an Oxymoron" in *Handbook of Organization Studies*, (Clegg, S., C. Hardy and W. Nord eds) Sage, London, pp. 440-458.

Wellman, B. (2001) "Physical Place and Cyberplace: The Rise of the Networked Individualism", *International Journal of Urban and Regional Research,* **1**.

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

Wetherell, M. (2001) "Debates in Discourse Research" in *Discourse Theory and Practice: A Reader*, (Wetherell, M., S. Taylor and S. J. Yates eds) Sage, Thousand Oaks, CA, pp. 380-399.

Wichmann, T. (2002a) "Basics of Open Source software markets and business models." *International Institute of Infonomics, Berlecom Research GmbH*

Wichmann, T. (2002b) "Firms' Open Source activities: motivations and policy implications" *International Institute of Infonomics, Berlecom Research GmbH*

Wichmann, T. (2002c) "Use of Open Source Software in Firms and Public Institutions. Evidence from Germany, Sweden and UK" *International Institute of Infonomics, Berlecom Research GmbH*

Wills, S. (1994) "2001: A Research Odyssey Teaching Different Types of Learning", *Journal of Management Development,* **13 (1),** pp. 60-72.

Winner, L. (1986) "Do Artifacts have Politics?" in *The whale and the reactor: a search for limits in an age of high technology.*, (Winner, L. ed.) University of Chicago Press, Chicago, pp. 19-39.

Winner, L. (1993) "Upon Opening the Black Box and Finding it Empty: Social Constructivism and the Philosophy of Technology", *Science, Technology and Human Values,* **18 (3),** pp. 362-378.

Winner, L. (1994) "Reply to Mark Elam", *Science, Technology and Human Values,* **19 (1),** pp. 107-109.

Wood, L. A. and R. O. Kroger (2000) *Doing Discourse Analysis: Methods for Studying Action in Talk and Text,* Sage Publications, Inc, Thousand Oaks, California.

Woolgar, S. (1991) "The Turn to Technology in Social Studies of Science", *Science, Technology and Human Values,* **16** pp. 20-50.

Wynn, E. (1979) Office conversation as an Information Medium.

Zeitlyn, D. (2003) "Gift economies in the development of open source software: Anthropological reflections", *Research Policy,* **Special Issue on Open Source Software Development.**

Zhang, W. and J. Storck (2001) "Peripheral Members in Online Communities". in *AMCIS,* Boston, MA,