Internet, Innovation, and Open Source: Actors in the Network

Ilkka Tuomi

SITRA

The Finnish National Fund for Research and Development[1]

ituomi@uclink4.berkeley.edu

November 3, 2000[2]

---

[1] The author is currently visiting scholar at the University of California, Berkeley, on leave of absence from Nokia Research Center.

[2] This paper is an edited version of a paper that was presented at the Association of Internet Researchers conference, Lawrence, Kansas, September 15, 2000.

Title: Internet, Innovation, and Open Source: Actors in the Network

Abstract:

This paper describes the evolution of the Linux operating system, and studies dynamics of socio-technical change using Linux as a case example. Theoretical models of community-based practice and learning are combined with actor-network theory, and the characteristics open source development model are described using the introduced theoretical concepts. The paper analyses the growth and development of Linux and its development community, and shows how the development community evolves into an ecology of community-centered practices.

Introduction:

During the last couple of years, the Open Source development model has been on front pages of newspapers, and focus of much attention (e.g., DiBona, Ockman, & Stone, 1999 Wayner, 2000; Leonard, 2000; Raymond, 1998b; Raymond, 1998a; Bezroukov, 1999; Kuwabara, 2000). It has been argued, for example, that open source projects can produce better quality technology than traditional corporate R&D (Raymond, 1999). As a result, many corporations have invested heavily in trying to adopt best practices from the open source model.

A distinctive characteristic of open source projects, when compared with traditional corporate software development projects, is the way intellectual property rights are handled. One key innovation in open source has been the GNU General Public License (Stallman, 1999), which has made it possible to legally improve and adopt software developed by others, at the same time facilitating continuous improvement.

In the history of software, open source models have, however, been used before copyrights became an issue. For example, it has been estimated that about half of the operating system programs for CTSS, an early time-sharing system at MIT, were developed by the users of the system (Fano, 1967). One of the motivations for launching the ARPANET project in 1960's was the belief that by connecting different computing sites, communities of computer programmers could more efficiently share their programs

and knowledge (David & Fano, 1965; Abbate, 1999; Naughton, 2000). Indeed, two of the most influential visionaries of ARPANET, J.C.R. Licklider and Robert Taylor argued in great detail in 1968 that such on-line communities would radically transform computer programming, but also society, work, and human thinking. Although they saw security and privacy as important challenges in on-line communities, their underlying assumption was that—within a given access control policy—software could be freely used and shared (Licklider & Taylor, 1968).

Copyright agreements should, therefore, be seen as one mechanism that has been used to solve a problem that is not fundamentally about intellectual property rights. Indeed, in open source development projects copyrights facilitate solving a well known problem. It is widely noted that access to source code facilitates learning, improvement, and integration with other systems. One important function of open source copyright agreements is that they keep this development path open, in a rapidly changing world where commercial appropriation of research and development investments has become increasingly difficult.

More generally, however, copyright is a social institution that helps social actors to coordinate and control their interactions. We can therefore ask what other mechanisms are used in open source projects, besides copyright, to facilitate innovation and development. This leads to a study on the co-evolution of social and technical systems.

This paper tries to develop theoretical understanding of the open source model. First, it briefly reviews proposals for conceptualizing the development of knowledge and technology in social and practice-related contexts. Then it introduces some central ideas of actor-network theory. Using data from the evolution of the Linux operating system, it then describes some key characteristics of the open source development model. The empirical findings will be interpreted in the light of the presented theory, and some conditions for successful open source development projects are suggested.

Thought communities: a brief review

During the last decade there has been increasing interest in understanding the social basis of technology and knowledge. It has been argued that knowledge exists only in a social

context, and that this social context is created by social practices. According to this view, knowledge is created and reproduced in communities, and knowledge makes sense only in relation to such communities. Furthermore, this view rejects the idea that knowledge can be decontextualized, or something that can in any trivial way be grounded on an "external reality." Instead, this view sees knowledge as a product of a social process. Knowledge organizes social by institutionalizing ways of interpreting the world. Knowledge is embedded in social practices, conceptual systems, and material artifacts that are used in social practices. Technology, social practice, and knowledge complement each other and their evolution is part of the same process.

Although such conceptualizations of knowing has gained visibility during the last years, their key ideas are not new. Bakhtin argued in the 1930's that speech and texts can only be understood by analyzing genres (c.f. Morson & Emerson, 1990). According to Bakhtin (1987), genres are created in a historical process where concepts, their use, and their practical context co-evolve. A competent adult has a large repertory of genres that are used in different concrete situations. Discussion with family members, lectures, private letters, academic manuscripts, and formal documents all have different genres. Utterances within different genres can look and sound similar, but their meaning can only be understood by analyzing their role within a specific genre. Each genre, furthermore, has its associated social settings and practices, and the evolution of a genre therefore is closely linked with the evolution of social practices and, for example, with those tools that are used in these practices. The constitution of a genre therefore has both "mental" and "material" components. Creative work, according to Bakhtin, requires effective use of those cultural resources that form the genre.

Ludwik Fleck proposed a similar view, also in the 1930s, based on his studies of historical development of syphilis as a specific disease entity. Fleck (1979) showed that scientific facts emerge through a long historical process, which produces interdependent theoretical conceptualizations, diagnostic practices, and technologies. According to Fleck, such conceptualizations, practices, and technologies are produced and reproduced

by thought communities. Each thought community has its own thought style, which defines what can be meaningful knowledge for the community in question.[3]

More recently, Donald Schön (1983) described learning processes that underlie acquisition of professional skills. According to Schön, competent professional practice can be understood as reflective practice, and such practices are learned and reproduced in communities of practitioners. Learning a professional skill is often based on social interaction, and competent use of appropriate technologies. Schön, for example, argued that an architect studio is a critical resource in learning architectural design as many key skills are tightly bound to the tools and material artifacts available in the studio. To became a member of the community of architects requires ability to learn to view the world as an architect and to use architect's tools in a professional way. This can happen only by observing and interacting with experienced architects within the context of a studio.

Yrjö Engeström (1987), in turn, developed his theory of activity systems and expansive learning on the basis of cultural-historical activity theory (Vygotsky, 1986; Leont'ev, 1978; Wertsch, 1991; Cole, 1996; Scribner, 1997; Engeström, Miettinen, & Punamäki, 1999). Cultural-historical activity theory argued that social practice should be understood as tool-mediated activity. Activity itself becomes meaningful only through sociocultural evolution, and therefore all meaningful human activity is inherently social. According to Engeström, learning new practices requires expansion of existing activity, and this, in turn, creates conflicts between the various interacting systems of activity, as well as between the old and new forms of activity within a specific community. For example,

---

[3] Fleck, himself, used the concept of thought collective. Indeed, Brown and Duguid (2000b) note that part of the recent enthusiasm around the community perspective may be due to the appeal of the word community. They point out that there would perhaps be less enthusiasm if Lave and Wenger (1991), who popularized the concept of community of practice in the early 1990's, would have used *cadre* or *commune*, instead of community. Fleck also used the concept of "thought style," which was later picked up by Mary Douglas (1987; 1996).

when one activity system produces tools that mediate the activity of another activity system, changes in one activity system may require change in the other.

Lave and Wenger formulated the activity-theoretic view on learning in a cultural-anthropological context, and proposed that the focal unit of social learning should be understood as a community of practice. According to Lave and Wenger (1991; Wenger, 1998), knowledge is learned by becoming a legitimate peripheral participator in a community of practice, and by gradually acquiring knowledge and reputation through a process of social interaction. Lave and Wenger also argued that learning is fundamentally about becoming an accepted member in the community. Expertise, identity, and membership in a community of practice are therefore inseparable.

Starting from studies on socio-technical evolution, Edward Constant (1987; 1984; 1980) argued that communities of practice are the loci of technological practice. According to Constant, communities of technological practice can consist of individuals or organizations. Technology usually develops through incremental improvement within an existing community of practitioners, but sometimes the community faces problems that require radical innovation. Traditions of technological practice, and the associated communities of practice, also rely on higher-level traditions of testability, including their accepted tools, procedures and values. Incremental innovation takes these boundary conditions as given, but sometimes radical innovation requires completely new kinds of systems of measurement and testing. Through such "higher-level" traditions, specific technological communities of practice become connected with normative engineering culture. In this way, according to Constant, "communities of practitioners reify the meaning of their tradition of practice for themselves and explain and justify that tradition to outsiders" (Constant, 1987:227).

Constant also proposed that technology should be seen as a form of knowledge. Similarly, Knorr Cetina (1999) has proposed that scientific practices can be understood as "epistemic cultures" that bind together tools, knowledge, and specific knowledge production mechanisms.

The idea of communities of practice has recently attracted much interest also in the context of organization and innovation theory (e.g., Sawhney & Prandelli, 2000; Brown & Duguid, 2000a; Kuusi, 1999; Tuomi, 1999b). Brown and Duguid (1991; 2000b) proposed that learning and innovation in organizations occur in and between communities of practice. Tuomi (1999a) combined the community of practice literature, cultural-historical activity theory, and Nonaka and Konno's (1998; Nonaka, Toyama, & Konno, 2000) model of knowledge creation "spaces" or "ba's," and proposed that organizations can be understood as "fractal communities" of interlinked systems of activity.

All these theoretical proposals guide the analysis of data in the present paper. Their common characteristic is that they focus on community of people as a locus of innovation, and argue that knowledge, practice, and technological artifacts are interdependent parts of an evolving social system. This concept of community, therefore, differs from those conceptualizations that view communities as groups of people. Instead, community is seen here as something that does not emerge from putting together a sufficient number of individuals. On the contrary, individuals became persons with individual identities through their membership in the various communities they are members of. Identity, in other words, is not something that is grounded on any possible list of "attributes" of an individual person. Instead, it is grounded on communities, with their specific systems of activity and collective meaning processing.[4]

---

[4] It should be noted that the different conceptualizations of practice-related communities, presented above, highlight different aspects of the phenomenon. The models of community are not usually developed in any great detail in the extant literature, and often the concept is used in somewhat ambiguous and even contradictory ways. There seems, however, to be four types or aspects of community around which the literature revolves. We could characterize these as "communities of production," "communities of interpretation, " "communities of identification," and "communities of appropriation." These are, of course, tightly interlinked, and difficult to separate in conceptual discussions, in empirical observation, and in practice. In the present paper I follow Fleck and use the term "thought community" to refer to all these. Although "thought community" is easily understood as a purely mental phenomenon, for example as a scientific paradigm in Kuhn's (1970) sense, Fleck's discussion on such communities included detailed

In such a context, open source development model is not only producing software. It also produces the interacting system of knowing, learning, and doing that organizes the community and its relations with other communities. Indeed, as the empirical analysis below shows, the open source development model is a heterogeneous network of communities and technologies. A characteristic of this model is that, under suitable conditions, technology development can become extremely rapid.

Actor-network theory and reduction of complexity

To describe characteristics of the open source development model, it is useful to introduce some key concepts of the actor-network theory. According to actor-network theory, society consists of networks of heterogeneous actors, both human and non-human (Latour & Woolgar, 1986; Bijker & Law, 1992; Callon, Law, & Rip, 1986; Latour, 1999). As the actors in the network can be both human and non-human, actor network theorists sometimes use the term *actant* to refer to such actors. Society, organizations, agents, and machines are all effects generated through the interactions of actor-networks. A person, for example, can not be understood as an isolated entity; instead, he or she is always linked to a heterogeneous network of resources and agents that define the person as the specific person in question.[5] Without his or her instruments, laboratory, and social relationships, a scientists, for example, loses his or her identity as a scientist.

Actor-network theory originated in studies of scientific practices, but it has become a generic framework for understanding social phenomena. A scientific laboratory may be viewed as a network of test tubes, diaries, scientific publications, budgets, and researchers, each with their own "competences" and "resistances." Scientific knowledge is produced in this network, and becomes an actor itself through new conceptualizations and observations recorded in journals, or, for example, by becoming embedded in

---

analysis of interdependencies between technologies, interpretations, identities, knowledge production, and their historical co-evolution.

[5] In this sense, also the intensional networks studied by Nardi, Whittaker, and Schwartz (2000) can be viewed as examples of actor networks.

scientific instruments and software code. A similar process underlies also evolution of other social institutions. Families, organizations, computing systems, the economy, and technology can all be similarly pictured (Law, 1992:381).

A key concept in actor-network theory is “translation.” The total system of actors in the full social network is extremely complicated. Reduction of this complexity is therefore a necessary requirement for practical action. Translation means a process where complicated sub-networks become represented by actants, and by which the complex underlying structure becomes a “black box” for practical purposes. For example, sometimes we can talk about “the British Government” without having to know what are its exact processes and who are the people that constitute it. Similarly, an organization can be represented by a single individual, and a complex system of accounting procedures can be represented by a software package.

Translation means that complex sub-networks become “punctualized,” and start acting like a unified entity, from the point of view of those actors who interact with the sub-network. At the same time such translated sub-networks become resources. For example, an existing scientific instrument can be used without considering all those processes, knowledge, and other resources that are required to manufacture it. Translation therefore means that complex networks can be taken for granted. But at the same time it means that the point of translation also becomes a locus of power and control. The effects produced by the translated sub-network become resources that can be located and controlled. Through this process of translation the punctualized network can be represented as if it were owned by the actor who manages the translation.

According to actor-network theory, the ongoing processes of translation are key sources of social order. Translation generates ordering effects, such as organizations, institutions, devices, and agents. Each of these have their own resistances, and social change therefore is very much about a struggle of reorganizing the resources and relations in the actor-network. In this process, resistances are anticipated and various strategies are deployed to overcome them. There is a continuous threat that existing order breaks down, and the fact

that order exists indicates that—at least in some pragmatic sense—strategies and translation processes work and form a relatively stable system.[6]

Evolution of Linux development resources

At this point we have two complementary proposals for understanding evolution of socio-technical systems. The community-based view argued that knowledge, technology, and learning occurs in practice-related communities, and that practices are embedded in material and technological artifacts. In such a context, learning both socializes community members, as Lave and Wenger noted, and creates new forms of activity and new products, as Engeström argued. Actor-network theory, in contrast, argued that human and non-human actors are symmetrical, and that they can often be replaced with each others. The key idea was that the complexity of sub-networks can be reduced by translation, which makes one actant able to stand for a whole sub-network.

Putting these two perspectives together shows how both these approaches can be refined and used to describe evolution of socio-technical systems, such as the Linux kernel. The tools used in social practice are translations of complex sub-networks that produce the tool, while simultaneously producing themselves as carriers of knowledge and related practices. As long as technology doesn't break down, its users can use technology as a tool. Such an object is effectively a black-box that mediates user's activity, without requiring the user to consider all the complex relationships that actually are hidden inside the system that makes the tool an object. For example, as long as everything works fine, a computer user doesn't have to know about electrical or digital design, or program

---

[6] Niklas Luhmann (1995) based his theory of social systems on a related idea. According to Luhmann, both meaning and social order emerge because complexity needs to be reduced. Meaning and social are, therefore, built from "black-boxes" that reduce contingency in a potentially extremely complex world. Meaning, for example, can be defined as order that emerges when one actual interpretation becomes selected from many possible "latent" interpretations in the cognitive process. The underlying order that makes the world a "meaningful" world is a network of meaning relations that provide the basis of interpreting the world. Similarly, the specific order that makes fundamentally contingent communicative interaction understandable is what we can define as "social" (c.f., Tuomi, 1999a).

architectures, any more than he or she needs to know how these things are developed and produced in practice, or where to find the experts that do know what is inside the box.

In this process of “black-boxing” sub-networks, translation processes do not only hide complexity of material components. Black-boxing also hides social networks and discourses. However, there are several different ways by which translation can be accomplished. If the black-box is represented through a material artifact, the black-box can be viewed as a “tool.” If it is represented by a human, the black-box can be viewed as an “organization.” If the translation process produces a mental product, the encapsulated network can be viewed as a “concept.”

We can illustrate this point by a simple outline of the history of the development of the Linux operating system.

Histories of Linux usually tell us that Linux was born when Linus Torvalds developed and distributed the first version of it, in 1991. In many ways, Linux, of course, is still work under progress, and constantly refined. But it also didn’t emerge from vacuum. The possibility of Linux was based on many earlier developments. Linux development process, for example, relied heavily on the existence of the Unix operating system, especially its BSD and Minix variants, newsgroups and listservers on the Internet, the GNU c-compiler and its libraries, and the GNU General Public License. Before Linux development started as a collaborative effort, many technology and knowledge creation communities had already been translated as resources for it.

Some critical actors that were used to produce resources for the Linux development community are show in Figure 1. Some of these actors are communities that can be described as “organizations” or “social networks,” others are material tools and conceptualizations produced by communities. For example, in the early phase of the development of packet-switched computer communication protocols the relatively informal Network Working Group (NWG) discussed potential uses of computer networks and developed the first specifications for the host-to-host protocols. The results of these discussions were later documented in Request for Comments (RFCs), which were first distributed in paper format, and later using the Arpanet itself (Braden, Reynolds, et al.,

1999). The Request for Comment –mechanism played a similar role in the development of Arpanet and Internet as source code in open source projects (Bradner, 1999; Naughton, 2000; Abbate, 1999).

Figure 1 shows that there were several important communities that produced resources that made it possible to start Linux development. It is impossible to discuss in detail the nature of these communities in this context. [7] To highlight the different types of actors in Figure 1, one can however note that they include organizational actors, such as ARPA / IPTO, i.e., ARPA's Information Processing Technologies Office which concentrated both visionary leaders and money; technological artifacts, such as ARPANET that was instrumental in the development of concepts and tools for distributed collaboration; and conceptual artifacts such TCP/IP protocol definitions that were documented in Request for Comments documents. The only business organization that is shown in the figure is Bolt, Beranek and Newman (BBN), the firm that developed the interface message processors for the ARPANET under a contract from ARPA / IPTO. Whereas some other organizational actors were business firms, they mainly acted as passive structures that were appropriated by the actual actors, for example, by the developers of Unix in AT&T Bell Labs.

[7] An interested reader can find details of the different communities from Abbate (1999) and Naughton (2000).

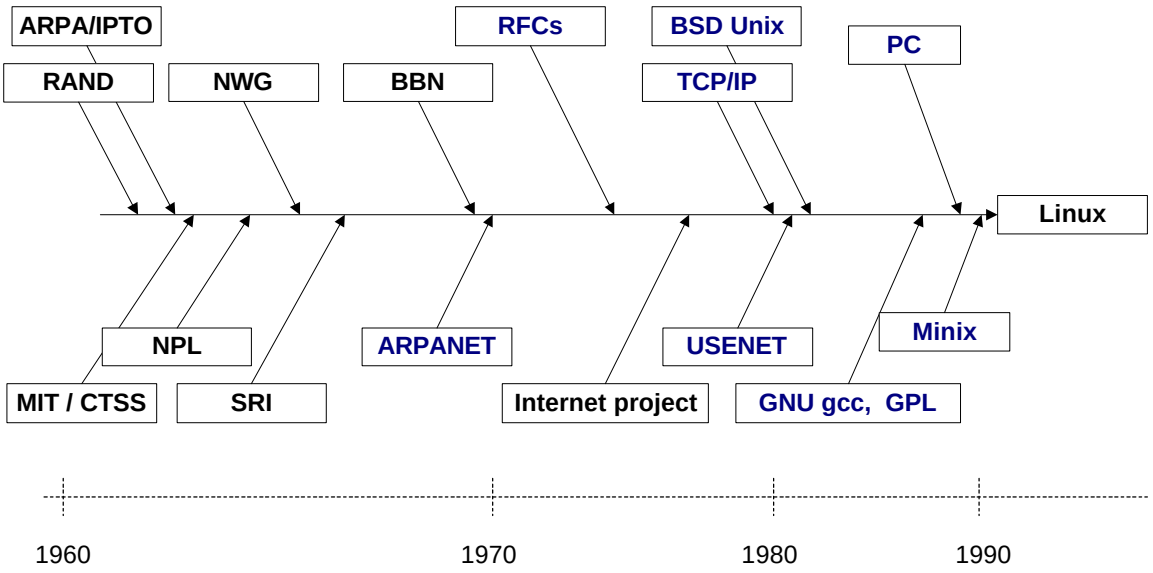

Figure 1. Key resource producing actors in early Linux development.

Although Figure 1 may give an impression of causality and inevitable succession of events, the evolution of the actor-network, of course, was not directed by any anticipation of the future success of Linux. Change in such a network should be viewed as a gradual movement of its different actants. The direction of evolution in such a system happens in directions where movement is fastest. Therefore, there is usually more than one way the aggregate network develops. For example, the resources shown in the figure did not only act as resources for Linux developers, but for many other Internet-enabled communities as well.

Growth of Linux

This brief theoretical discussion now enables us to describe the evolution of Linux and its developer community. When the development of Linux source code started in 1991, the existing resources enabled very rapid growth. This growth is still going on, as can be seen in Figure 2. The core operating system, Linux kernel, has been expanding almost exponentially. This is rather remarkable as a high-quality operating system kernel requires that the code is optimized for speed and as the collaborative development mode means that there is a strong priority of producing as simple source code as possible. The growth of source code, therefore, is not generated by adding new features to the kernel in

a random fashion. Instead, as will be shown below, the growth comes from a highly organized expansion of the kernel.

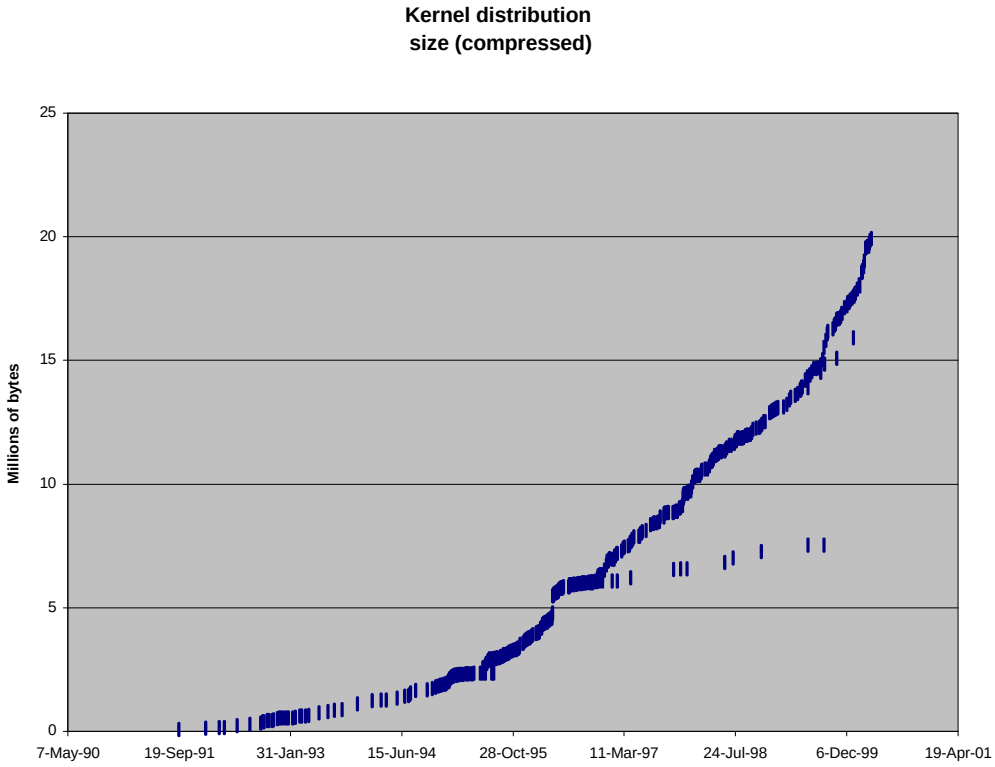

Figure 2. Growth of the Linux kernel.

Linux developer community

From the very early history of Linux, its development process has been collaborative. A special characteristic of this collaboration has been that it has almost completely relied on Internet-based tools. In contrast to many arguments that effective virtual collaboration requires existing social interactions, the Linux developer community has been almost completely virtual. Furthermore, it started as a virtual community. As a result, the community was able to attract members who were geographically distant.[8] This can be

[8] Raymond and others have argued that open source is based on post-scarcity economy and abundance of resources. The evolution of the Linux development community shows, however, that this is not the whole

seen from Figure 3. The figure shows the distribution of key Linux developers in different countries, per million inhabitants. The data is based on analyzing the first CREDITS –file that recorded key authors in 1994.[9]

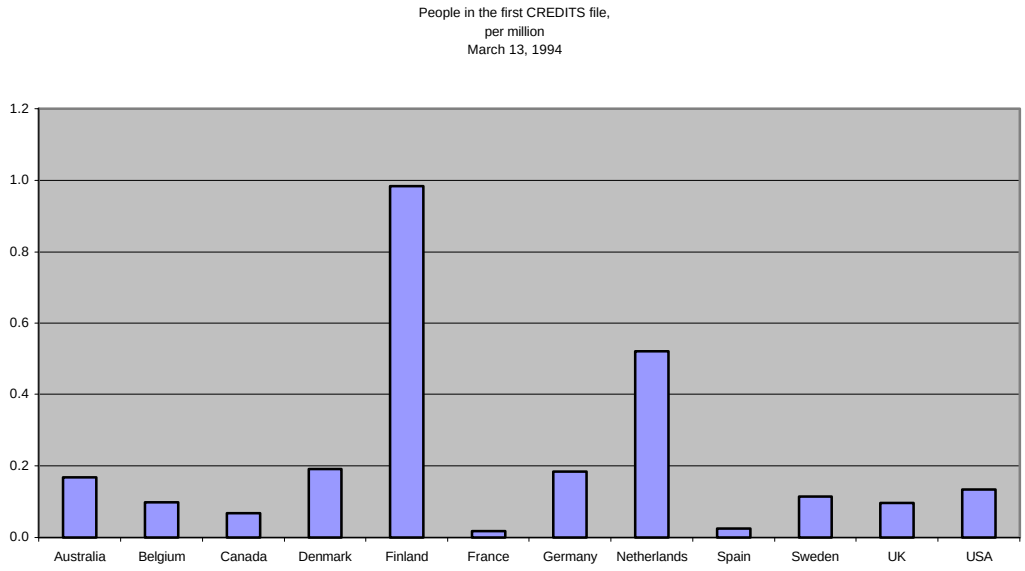

Figure 3. Early key developers in different countries.

Sedimentation of source code

In the early years of Linux development, its source code was mainly used as a platform for further development of the code itself. When Linux started to be a viable operating system, it became used by people who can be characterized as "end-users." For such end-

---

story. For example, according to Torvalds an important reason to distribute Linux through Internet was the scarcity of development resources in the Helsinki University (personal communication, September, 2000). In this sense, the early phases of Linux development were very similar to the early phases of the WorldWideWeb (Berners-Lee & Fischetti, 1999). As Castells (2001) has noted, resource scarcity also promotes adoption of open source in countries where resources are limited.

[9] I have described the history of Linux and its developer community in detail in a forthcoming book (Tuomi, 2001).

users, Linux was not a complex system of interacting source code modules and programming tools. Instead, Linux became a resource. Furthermore, Linux distributors bundled the operating system kernel with applications and utility programs, and effective distribution required efficient management of software configurations. This created a tension in the Linux development model. For some user-developers Linux was a system where new components were frequently added and which provided interesting opportunities to make novel and high-impact contributions. For such users, Linux remained a complex and evolving network of software modules, function calls, and software procedures. For others, this flexibility was a problem. Continuous change intervened with the translation processes and made it difficult to use Linux as a resource.

As a result of this tension between end-users and developers, the development of Linux has been split into two development paths. One path is a “stable” path, where only essential changes are introduced. The other path is a “development” path, with continuous integration of new and useful code components. The branching of source code paths can be seen in Figure 2, and in more detail in Figure 4. This process is interesting as it shows how fundamentally the same product can be translated into a resource and simultaneously keep evolving as a network. The need to translate Linux code into a resource for other communities produces a “sedimented” or “black-boxed” version of the code. In the terminology of actor-network theory, such forking of the development paths to create a black-box version of the system is a translation strategy that reduces the struggles between end-users and developers. In the stable path, the translation process itself can remain unchanged until a radically new version is produced and becomes the current resource.

Sedimentation is a good name for this phenomenon as typically several layers of sediments become formed during the evolution of a system. Such sediments, however, do not necessarily remain stable. The modularization of Linux code generates and ecology of development communities (Tuomi, 2001). Each new community articulates its resources and at the same time creates new tensions in the underlying networks. Eventually, these tensions may deform and break the existing structures.

Cultural-historical activity theory argued that human activity is always intentional and oriented towards an “object of activity” (Leont'ev, 1978; Stetsenko, 1995; Gal'perin, 1992). According to Leont’ev, such an object can be understood as the motive of activity. Leont’ev developed a model of human activity which was based on three levels of analysis: meaningful productive activity, its decomposition into goal-oriented actions, and further into operations that implement actions within a specific context. Leont’ev’s analysis of the development of activity in sociocultural evolution showed that there is a constant movement between the different levels in this analytical hierarchy. For example, goals can easily become motives. Whereas a group of hunters may understand the manufacturing of their hunting tools as goal-oriented action within the context of hunting activity, when social division of labor creates a community of tool manufacturers, for this community tool manufacturing becomes the object of their activity. Motives, community structure, productive processes, and resources used in these productive processes are therefore interdependent and dynamically changing.

In the context of Linux development, such movement of motives is clearly visible. In the early phases of the development, the object of activity was the Linux kernel itself. When Linux was robust enough so that it could be appropriated for application development, it became a tool for application development communities. Finally, when Linux became used as an operating system to run applications, a full GNU/Linux distribution become a tool that was combined with hardware and closed into a box.

Whereas Linux kernel developers need relatively open access to the Linux source code, and application developers may greatly benefit from such open access, for end-users it is relevant mainly in those cases where the black-box breaks down and reveals its true nature as a complex system of actors in a network. The proposed “superiority” of the open source model, therefore, to a large extent reflects the fact that computer systems often *do* break down. The value of open source approach is, however, greatly reduced if the end-users do not have enough competences to diagnose the sub-network that becomes visible when the translation stops. The relatively good scalability of the open source model, therefore, seems to result from the fact that it also facilitates competence development. More generally, transparency of the underlying system makes it possible

for the end users to mobilize all resources and competences they have available to solve the problem at hand, including those that no-one had thought before. The specific "style" in which open source systems break down therefore promotes effective use of problem solving resources, at the same time facilitating development of competences that can be used in solving similar problems in the future.

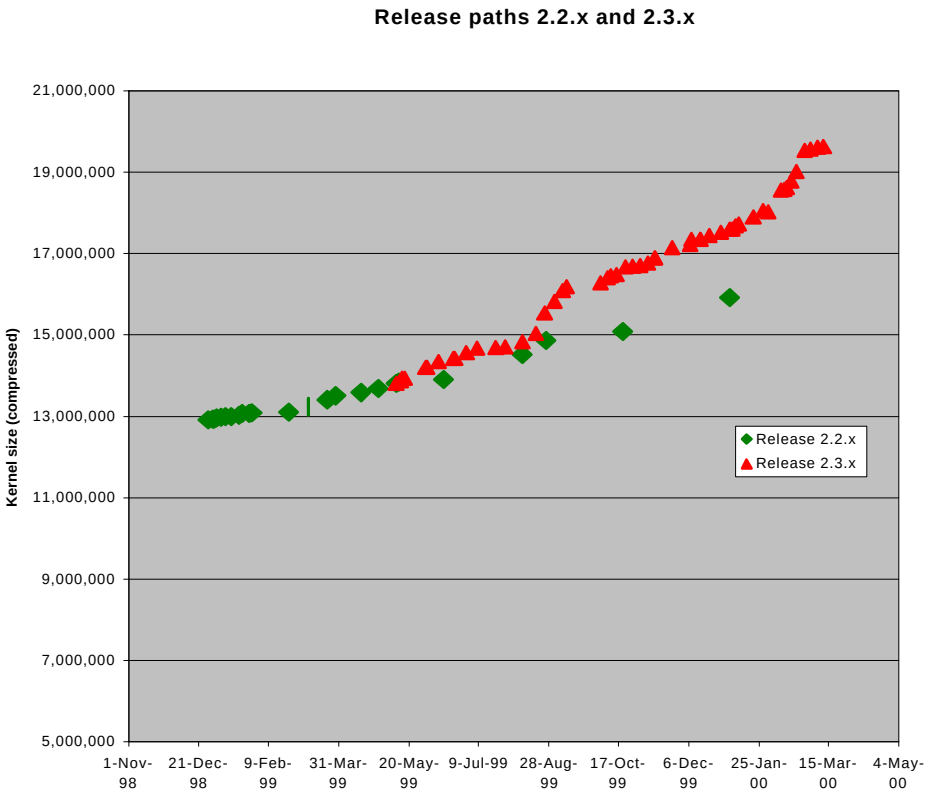

Figure 4. Resource and object development paths.

To extend the metaphor of sedimentation: open source implies that layers which become sedimented remain soft. If there is a problem, it is relatively easy to dig one's way through the module interfaces to see where the problem is and how it can be corrected.

Structural evolution of the Linux architecture

As was noted above, for its developers a computer operating system is a complex network of modules. In the course of the evolution of the system, functionality of the system becomes abstracted, and a homogenous mass of computer code gets divided into

relatively loosely coupled components that interact with each other in relatively well defined ways. Indeed, modularization is commonly seen as a key to effective software development. The system under design is decomposed into some natural components, which can then be implemented by programmers. Often each module is assigned a team of programmers who are responsible for developing and maintaining the module.

In the evolution of Linux, software architecture and the organization of its developer community are not based on a given conception or design of the system. Instead of purely functional considerations, the Linux architecture reflects fundamental social problems of coordinating and mobilizing resources. Although both the existing abstractions for the Unix architecture and the existing microprocessor hardware architectures constrain the way the developer community can effectively be organized, the Linux architecture reflects also to a considerable extent requirements of collaborative development.

One way to observe the internal network of actors in Linux development is to study the modularization of Linux source code. The basic heuristic in modularization is to put in one "place" source code that can be developed as an independent entity. In the case of Linux, the place is a file directory that stores a file or a group of closely related files. Although there are exceptions, and a close mapping between modules and directories sometimes breaks down, as a first approximation it is possible to study the evolution of Linux architecture by studying the evolution of these directories.[10]

The size of files in the different Linux kernel directories for two kernel releases are shown in Figure 5. As the figure shows, in the course of Linux development new modules are added, some old modules are removed, and the speed of growth varies considerably between the different modules. The developers of Linux need to know what these directories contain, and which of them are important for the developer's present activity.

---

[10] The following discussion is based on a comparative study of documented Linux and Unix architectures, concrete architectures generated by automatic architecture extractors, and detailed studies on the evolution of the kernel source code files (Tuomi, 2001).

A source code module often acts as a punctualized resource. One does not need to know the exact implementation details to develop code that interacts with a given module. The module defines an interface that can be used to interlink with the module. This interface translates the technical system and the community that develops and maintains it, so that it becomes a resource for another community. A standard procedure—often implemented as an "interface"—is used to access the services provided by the resource. As long as the protocol for using the resource and the service associated with it are not changed, the users of the interface don't have to know the internal details of the technological artifact or the organization of its production network.

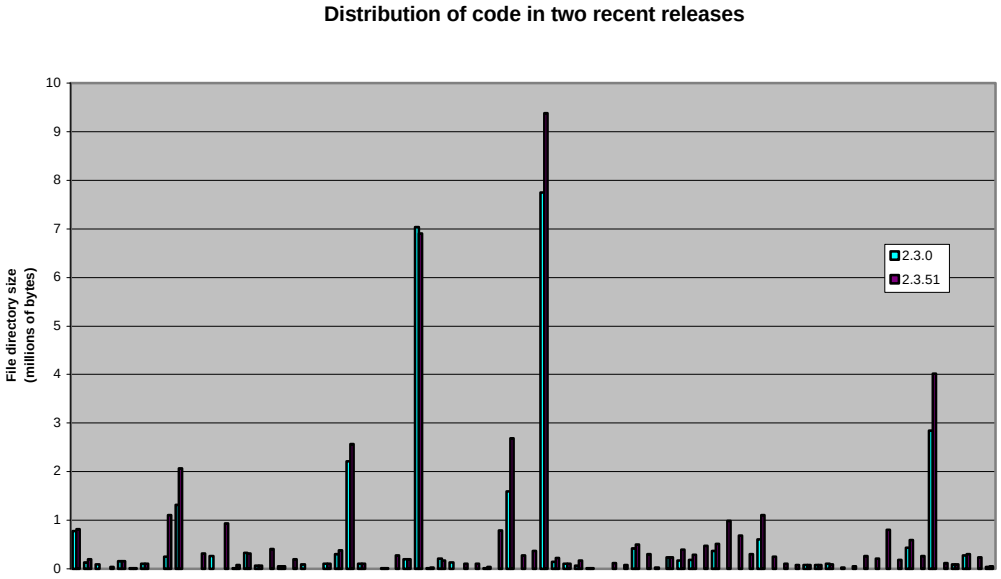

Figure 5. Creative destruction in the Linux kernel.

According to Schumpeter (1975), innovation produces creative destruction. The fundamental characteristic of the capitalistic socio-economic system is that resources are dynamically moved from old uses to new ones. Opportunities are materialized in an entrepreneurial process, and the speed of innovation depends on the speed of moving resources from old activities to new ones. Figure 5 shows that such a process also occurs at the level of technological artifacts. Some modules disappear in the course of the evolution, and their developers move to new activities.

A closer analysis of this process reveals, however, that Linux has several qualitatively different “regions of innovation.” A similar process of sedimentation that was seen on the level of Linux kernel can also be seen inside the kernel. The end-users want to use Linux as a resource without the need to consider the complex network of its various modules that were reflected in Figure 5. Similarly, the developers of Linux kernel need to simplify the complexity of the development network. Specifically, almost all module developers rely on some key components of the system. The translation processes for these key components have to translate the underlying sub-networks simultaneously for many different actors. This is accomplished by sedimenting the resource. In other words, the potential problems of maintaining a complex network of changing translation processes is solved by standardization of the translation process and by stopping development that could break the black-box. This can be seen in Figure 6. The figure shows the change in code size in the components of the core Linux kernel. This “hard core” encapsulates the nucleus of Linux kernel so that Linux developers can continue working on other parts of the system.

The fact that development in these core components slowed down very quickly in the evolution of Linux indicates how difficult it is to provide multi-faceted translation interfaces. One might read Figure 6 as showing that when several different actors approach a sub-network each from their own perspective, no common abstraction is good enough. In other words, there is no generic packaging for changing black boxes. Instead, the code has to be frozen as a concrete technical artifact.

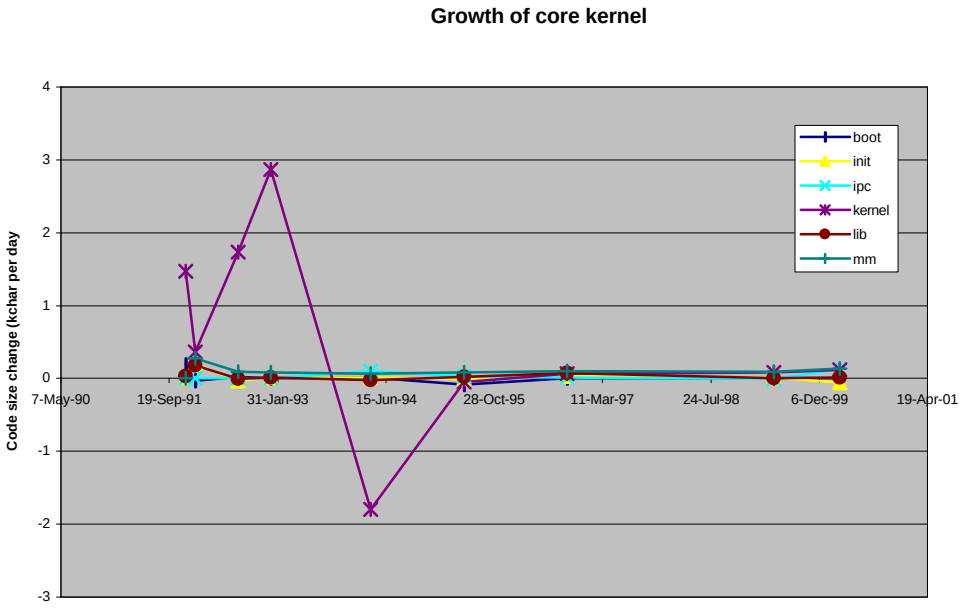

Figure 6. Rapid stabilization of core components.

The "core" of Linux kernel in Figure 6 is defined as those components that have stabilized in the early phases of the development process. This conceptualization means that there is no predetermined categorization of the components, for example, based on theoretical understanding of what are the "foundational" layers of a typical Unix operating system architecture. Instead, the "foundational" components are defined as those components that provide a foundation. The fact that these components acquire this role depends very much on the fact that the translation of the foundation has to address the needs of several actors. In this sense, foundational components of the structure are its "institutional" components.

In the Linux architecture, institutional innovation seems to be rare, and the slow change in the source code seems to be related to the problems of translation. Some other parts of the Linux architecture, however, grow very rapidly. Such rapidly growing components are shown in Figure 7.

The most rapidly growing part of the Linux architecture is a set of device drivers. When new hardware is introduced, Linux developers very quickly integrate it with the Linux operating system. Indeed, Linux development is to a large extent organized as "projects"

that focus on adapting specific hardware products into the Linux actor-network by "gluing" them to the system with software. Linux, therefore, can be viewed as an actor that quickly appropriates new technological elements and turns them into resources for the Linux user community. This is also probably the main difference between conventional software projects and the Linux development project. Linux is clearly an ecology of socio-technical development, not a project that implements a predefined plan.

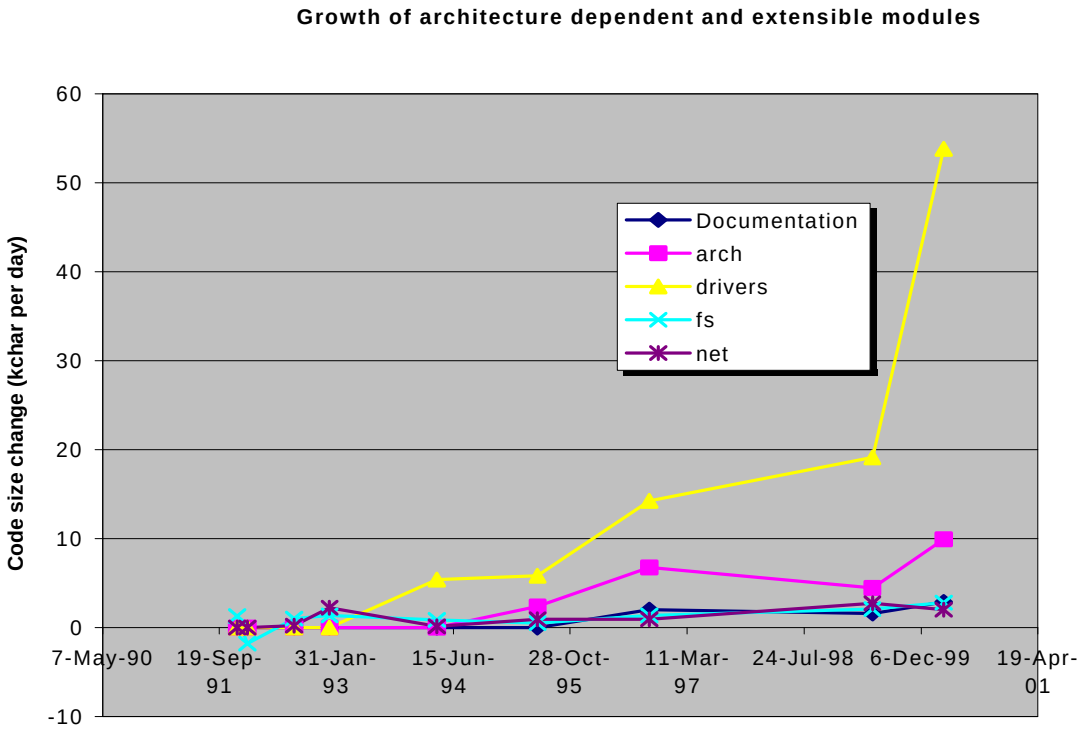

Figure 7. Continuous development in the periphery.

Proliferation of resources and actants

In the course of Linux evolution, many new translation mechanisms have been invented. Linux is an exceptionally interesting case of such proliferation of translation mechanisms as Linux software developers are able to create technical solutions to the problems of translation. In this sense, the Linux community is not only a Linux developer community, but also a tool-developing community. Indeed, one could argue that this is one of the reasons why Linux development has been rapid. The boundaries between Linux

development activity and Linux tool development activities can be crossed rapidly and without great effort.

This strength of Linux development model has been inherited from the Unix culture. Unix was developed with the idea that it would be a set of tools that can be easily combined and reused as components of new tools. The Linux developer community, therefore, should not only be viewed as a community that develops the Linux kernel. The success of its development model is critically dependent on resources that it appropriates and develops for the various tasks of the community.

This ecology of resources is a complex one, and a main challenge in becoming a competent Linux developer is to learn how to use these resources. Some of the resources can be characterized as organizational or community resources, others can be viewed as technological artifacts or tools, and as information resources.

Open source literature has very much emphasized the capability of open source projects to produce reliable and bug-free software, and argued that this is the key difference between open source projects and traditional software development projects. Some important resources used in the Linux bug-removal process are shown in Table 1. The table categorizes the resources as information resources, tools, and communities. Information resources are texts that can be used to learn what the community is doing, what its practices are, and what are the resources available for it. Tools are resources used in the actual bug-removal practices. Community resources are used keep the community alive and coordinate its activities. As the table shows, one technological artifact can have multiple roles in this ecology. For example, the JitterBug system is a web-based database which shows what bugs are known to the community and whether someone is doing something to correct a bug. JitterBug acts as an information resource by allowing people to find out what bugs are known, and as a community resource by coordinating the work needed to solve the problem.

The main actants in the bug removal process are shown in the table below. These actants can be viewed as resources that translate the underlying sub-networks.

processing phase			information resource	tool	community resource
detect			compiled code documentation	man	LDP
debugging	characterize		source code linux-kernel list FAQ JitterBug oops-tracing.txt Kernel Traffic LDP project-specific sites linux-kernel archives README files log files bug reporting form	editor gcc make gdb ksymoops IRC computer configuration	linux-kernel list JitterBug personal email IRC channels kernel-newsflash LDP project-specific lists
	remove		source code	editor gcc make	
		test	patch MAINTAINERS file	diff gcc make editor ftp	personal email linux-kernel list
distribute			patch MAINTAINERS file	gzip tar email ftp	linux-kernel list JitterBug
integrate			patch release	CVS vger package managers	Maintainers vger

Table 1. Actors in the Linux bug removal process.

In the evolution of complex system of resources and communities, social organization and tools co-evolve. New technological artifacts are created by groups of people who organize their work around the development of the artifact. A new artifact, therefore, creates a new community in the context of the originating community. This process leads to increasing differentiation in the social system.

At the same time resources that are produced outside the focal community are appropriated by the community. For example, the c-language compiler produced by the GNU gcc community is generated outside the Linux development community. The gcc compiler is, however, a critical resource for the Linux community (Torvalds, 1999). If the

compiler would become unavailable, it is probable that Linux development would become impossible.

The GNU General Public License obviously plays an important role here. It guarantees that the GNU gcc compiler can be appropriated by the Linux community as a core resource in the development. By relying on the institution of copyright, open source licenses provide an institutional basis for reducing risk and building knowledge based trust (Lewicki & Bunker, 1996). Without open source license, it would be very risky to build a system that so critically depends on a resource that is produced outside the community.

Indeed, open source licenses themselves can be seen as standardized translations which simultaneously provide multifaceted interfaces for many different actors. In very concrete terms, you don't have to negotiate licenses for open source: the license creates a universal standard interface that links the system to potential developers and users. This standardized interface limits the increase in complexity when new communities and actors start to use translated resources in their own activities.

By using the institutional basis of intellectual property rights to create a multifaceted translation interface, open source, however, also interfaces the system to the economic domain in a very specific way. An open source license makes the evolution of the system indifferent to economic values, as they have been conventionally understood. Although money may constrain open source development, open source licenses can often be read as statements that the real action is somewhere else than in the economic domain, and that money is an irrelevant measure of value in open source projects. In open source, economy, in other words, is an externality.

Discussion

Two complementary approaches to innovation and socio-technical change have been used above to interpret the evolution of Linux operating system kernel. First we argued that knowledge is located and develops in communities that are organized around practices. Knowledge is tightly linked to technologies used in these practices, and to the system of meanings which the community uses to communicate and make sense of the

world. This “community-centric” view has earlier been used to discuss creation of meaning and knowledge by Bakhtin and by Fleck, and more recently to describe social learning, by Schön and by Engeström, and more specifically, socialization to existing traditions and practices by Lave and Wenger and others. Actor-network theory, in turn, has been used to describe the evolution of socio-technical systems, often focusing on the struggles and strategies of appropriating and creating power in the network.

The present paper puts these approaches in a context of an ecology of communities and a modular technical architecture. Specifically, we have tried to show how the changes and dynamics of technological architectures reflect tensions that are created in developing the system in question. We have put actor networks inside communities of practice, and briefly described how communities become actors in a network of communities.

We have therefore also changed the way actor-networks and communities of practice have often been interpreted. The theory of actor-networks has the problem that it potentially makes humans and non-humans too symmetrical. It is as if machines, tools, and technologies would have their own motives and will in the same way as humans. This assumption, of course, would require careful discussion on the nature of motives, and here activity theory can bring useful insights (Miettinen, 1999). Such discussion, however, leads to a view that sees motives as grounded to social practice, division of labor, and tool mediated activity. The locus of activity can then be found in a community that organizes itself around the specific practice in question. By arguing that communities are special and fundamental types of actors in an actor-network we can describe what makes the evolution of actor-networks possible and how such evolution occurs.

On the other hand, by utilizing the concepts of translation, punctualization, and resource as described in the actor-network theory, we can better understand the evolution of practices and communities. This is important for understanding technological change, as new technologies are always appropriated by integrating them into social practice. Indeed, one can argue that innovation occurs *only* when social practice changes. Often such change results from appropriation of a new tool, which reorganizes the practices of a community. The key to innovation, therefore, is in those social communication and

learning processes that underlie change in social practices. The accumulated learnings, however, also become partly embedded and sedimented into the architecture of the technological artifact that is developed in the process.

Social practices, however, are interlinked in the ecology of communities. Resources and tools that are used in a given practice are produced in other practices. It is not always possible to change social practice without breaking those translations processes that make a community a resource to other actors. Change is difficult especially when the same translation process is used by several actors. As the evolution of Linux shows, one way to solve this problem is to sediment resources, institutionalize practices, and stop innovating.

The history of Linux, however, also shows that effective translation mechanisms can lead to rapid growth. The problem of managing interfaces between modules has lead to relatively standardized ways of building and using interfaces. This, in turn, means that modules can easily be added to the system. Furthermore, these standardized translation mechanisms mean that modules can be relatively easily used by different actors even when the modules change.

Linux is therefore in many ways open to combinatorial innovation. Standardized interfaces and translation processes generate smooth module boundaries and facilitate rapid recombination. The source code itself can be sometimes reused, but more importantly, the learning that is represented in the source code can be reapplied in different contexts without major problems. As a result, the various communities that develop the different parts of the Linux kernel become very mobile. In this way, the solution to the problem of translation leads to an ecology of communities that can readily reconfigure its resources.

In the Linux development, the Schumpeterian creative destruction destroys pieces of code, but competence and experience are reorganized with little waste. In this sense, one could argue that the Linux development model and the Silicon Valley innovation model (Kenney, 2000) have similar characteristics. As motives and values emerge and become articulated within specific social contexts, one could expect that the cultures of Silicon

Valley and Linux development communities are relatively easy to integrate without major conflicts. The main difference, of course, is that Silicon Valley has a venture capital driven entrepreneurial culture, whereas the economic sphere has been relatively invisible in the Linux development. This tension is actively being managed by the Open Source Initiative (c.f. Raymond, 1999). Indeed, the Open Source Initiative can be seen as one more organizational form or a community that springs up in the evolution of Linux to repair social damage that is created when these two relatively similar cultures collide and create conflicts in the developer community.

As the analysis of the evolution of Linux shows, rapid growth requires that the core is institutionalized and that some of the translation processes are taken for granted. In this model, innovation happens in periphery. It is interesting that such peripheries are conventionally described as frontiers. We could ask, however, whether—and in what sense—progress results from moving the boundaries of periphery, or whether this simply is one strategy to reduce change in the core.

References

Abbate, J. (1999). *Inventing the Internet.* Cambridge, MA: The MIT Press.

Bakhtin, M. (1987). *Speech Genres and Other Late Essays.* Austin, TX: University of Texas Press.

Berners-Lee, T., & M. Fischetti. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor.* San Francisco: HarperCollins Publishers.

Bezroukov, N. (1999). A second look at the Cathedral and the Bazaar. *First Monday,* **4** *(12)* http://firstmonday.org/issues/issue4_12/bezroukov/

Bijker, W.E., & J. Law. (1992). *Shaping Technology / Building Society: Studies in Sociotechnical Change.* Cambridge, MA: The MIT Press.

Braden, R., Reynolds, J.K., Crocker, S., Cerf, V., Feinler, J., & Anderson, C. (1999). RFC 2555: 30 Years of RFCs. The Internet Society. ftp://ftp.isi.edu/.

Bradner, S. (1999). The Internet Engineering Task Force. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open Sources: Voices from the Open Source Revolution.* (pp. 47-52). Sebastopol, CA: O'Reilly & Associates, Inc.

Brown, J.S., & Duguid, P. (1991). Organizational learning and communities of practice: toward a unified view of working, learning, and innovation. *Organization Science,* **2**, pp.40-57.

Brown, J.S., & P. Duguid. (2000a). *The Social Life of Information.* Boston, MA: Harvard Business School Press.

Brown, J.S., & Duguid, P. (2000b). Knowledge and organization: a social-practice perspective. *Organization Science,* in press.

Callon, M., J. Law, & A. Rip. (1986). *Mapping the Dynamics of Science and Technology: Sociology of Science in the Real World.* Houndmills, Basingstoke: The Macmillan Press Ltd.

Castells, M. (2001). *The Internet Galaxy: Reflections on Internet, Business, and Society.* Oxford University Press.

Cole, M. (1996). *Cultural Psychology: A Once and Future Discipline.* Cambridge, MA: The Belknap Press of Harvard University Press.

Constant, E.W. (1980). *The Origins of the Turbojet Revolution.* Baltimore: Johns Hopkins University Press.

Constant, E.W. (1984). Communities and hierarchies: structure in the practice of science and technology. In R. Laudan (Ed.), *The Nature of Technological Knowledge: Are Models of Scientific Change Relevant?* (pp. 27-46). Dordrecht: Reidel.

Constant, E.W. (1987). The social locus of technological practice: community, system, or organization? In W.E. Bijker, T.P. Hughes, & T.J. Pinch (Eds.), *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology.* (pp. 223-242). Cambridge, MA: The MIT Press.

David, E. E. Jr and Fano, R. M. (1965). Some Thoughts About the Social Implications of Accessible Computing. Excerpts reprinted in IEEE Annals of the History of Computing, 14 (2), pp.36-9.

DiBona, C., S. Ockman, & M. Stone. (1999). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, CA: O'Reilly & Associates, Inc.

Douglas, M. (1987). *How Institutions Think.* London: Routledge & Kegan Paul.

Douglas, M. (1996). *Thought Styles: Critical Essays on Good Taste.* London: SAGE Publications.

Engeström, Y. (1987). *Learning by Expanding: An Activity Theoretical Approach to Developmental Work Research.* Helsinki: Orienta Konsultit.

Engeström, Y., R. Miettinen, & R.-L. Punamäki. (1999). *Perspectives in Activity Theory.* Cambridge: Cambridge University Press.

Fano, R. M. (1967). The computer utility and the community. IEEE International Convention Record, 30-34 Excerpts reprinted in IEEE Annals of the History of Computing, 14 (2), pp.39-41

Fleck, L. (1979). *Genesis and Development of a Scientific Fact.* Chicago, IL: The University of Chicago Press.

Gal'perin, P.I. (1992). The problem of activity in Soviet psychology. *Journal of Russian and East European Psychology,* **30** *(4),* pp.37-59.

Kenney, M. (2000). *Understanding Silicon Valley: The Anatomy of an Entrepreneurial Region.* Stanford, CA: Stanford University Press.

Knorr Cetina, K. (1999). *Epistemic Cultures: How the Sciences Make Knowledge.* Cambridge, MA: Harvard University Press.

Kuhn, T.S. (1970). *The Structure of Scientific Revolutions.* Chicago: The University of Chicago Press.

Kuusi, O. (1999). Learning communities as sources of innovations and as targets of innovation policy. In G. Schienstock & O. Kuusi (Eds.), *Transformation Towards a Learning Economy: Challenges for the Finnish Innovation System.* Helsinki: SITRA.

Kuwabara, K. (2000). Linux: a bazaar at the edge of chaos. *First Monday,* **5** *(3)* http:// firstmonday.org/issues/issue5_3/kuwabara/

Latour, B. (1999). *Pandora's Hope: Essays on the Reality of Science Studies.* Cambridge, MA: Harvard University Press.

Latour, B., & S. Woolgar. (1986). *Laboratory Life: The Construction of Scientific Facts.* Princeton, NJ: Princeton University Press.

Lave, J., & E. Wenger. (1991). *Situated Learning: Legitimate Peripheral Participation.* Cambridge: Cambridge University Press.

Law, J. (1992). Note on the Theory of the Actor-Network: Ordering, Strategy, and Heterogeneity. *Systems Practice,* **5** *(4),* pp.379-93.

Leonard, A. (2000). Free Software Project. Salon.com: http://www.salon.com/tech/fsp/index.html.

Leont'ev, A.N. (1978). *Activity, Consciousness, and Personality.* Englewood Cliffs, NJ: Prentice-Hall.

Lewicki, R.J., & Bunker, B.B. (1996). Developing and maintaining trust in work relationships. In R.M. Kramer & T.R. Tyler (Eds.), *Trust in Organizations: Frontiers of Theory and Research.* (pp. 114-139). Thousand Oaks, CA: SAGE Publications, Inc.

Licklider, J.C.R., & Taylor, R.W. (1968). The computer as a communication device. *Science and Technology, (April)* Reprinted: In Memoriam: J.C.R. Licklider (1915-1990), Digital Systems Research Center, August 7, 1990. ftp://ftp.digital.com/pub/DEC/SRC/research-reports/

Luhmann, N. (1995). *Social Systems.* Stanford, CA: Stanford University Press.

Miettinen, R. (1999). The riddle of things: activity theory and actor-network theory as approaches to studying innovations. *Mind, Culture, and Activity,* **6** *(3),* pp.170-195.

Morson, G.S., & C. Emerson. (1990). *Mikhail Bakhtin: Creation of Prosaics.* Stanford, CA: Stanford University Press.

Nardi, B.A., Whittaker, S., & Schwartz, H. (2000). It's Not What You Know, It's Who You Know: Work in the Information Age. *First Monday,* **5** *(5)* http://firstmonday.org/issues/issue5_5/nardi/

Naughton, J. (2000). *A Brief History of the Future: From Radio Days to Internet Years in a Lifetime.* Woodstock: The Overlook Press.

Nonaka, I., & Konno, N. (1998). The concept of "ba": building a foundation for knowledge creation. *California Management Review,* **40** *(3),* pp.40-54.

Nonaka, I., Toyama, R., & Konno, N. (2000). SECI, ba, and leadership: a unified model of dynamic knowledge creation. *Long Range Planning,* **33** *(2000),* pp.5-34.

Raymond, E.R. (1998a). Homesteading the noosphere. *First Monday,* **3** *(10)* http://firstmonday.org/issues/issue3_10/raymond/, and http://www.tuxedo.org/-esr/writings/

Raymond, E.R. (1998b). The Cathedral and the Bazaar. *First Monday,* **3** *(3)* http:// firstmonday.org/issues/issue3_3/raymond/, and http://www.tuxedo.org/-esr/writings/

Raymond, E.R. (1999). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* Sebastopol, CA: O'Reilly & Associates, Inc.

Sawhney, M., & Prandelli, E. (2000). Communities of creation: managing distributed innovation in turbulent markets. *California Management Review,* **42** *(2),* pp.24-54.

Schön, D.A. (1983). *The Reflective Practitioner.* New York: Basic Books.

Schumpeter, J.A. (1975). *Capitalism, Socialism and Democracy.* New York: Harper & Row.

Scribner, S. (1997). *Mind and Social Practice: Selected Writings of Sylvia Scribner.* Cambridge: Cambridge University Press.

Stallman, R. (1999). The GNU operating system and the free software movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open Sources: Voices from the Open Source Revolution.* (pp. 53-70). Sebastopol, CA: O'Reilly & Associates, Inc.

Stetsenko, A.P. (1995). The role of the principle of object-relatedness in the theory of activity. *Journal of Russian and East European Psychology,* **33** *(6),* pp.54-69.

Torvalds, L. (1999). The Linux edge. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open Sources: Voices from the Open Source Revolution.* (pp. 101-111). Sebastopol, CA: O'Reilly & Associates, Inc.

Tuomi, I. (1999a). *Corporate Knowledge: Theory and Practice of Intelligent Organizations.* Helsinki: Metaxis.

Tuomi, I. (1999b). Inside innovation clusters: collective knowledge creation in networks and communities. In G. Schienstock & O. Kuusi (Eds.), *Transformation Towards a Learning Economy: Challenges for the Finnish Innovation System.* Helsinki: SITRA.

Tuomi, I. (2001). *Theory of Innovation: Change and Meaning in the Age of Internet (working title).*

Vygotsky, L. (1986). *Thought and Language.* Cambridge, MA: The MIT Press.

Wayner, P. (2000). *Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans.* New York: HarperBusiness.

Wenger, E. (1998). *Communities of Practice: Learning, Meaning, and Identity.* Cambridge: Cambridge University Press.

Wertsch, J.V. (1991). *Voices of the Mind: A Sociocultural Approach to Mediated Action.* Cambridge, MA: Harvard University Press.