The Linux Kernel Development As A Model of Open Source Knowledge Creation

Gwendolyn K. Lee
Robert E. Cole
Haas School of Business

December 2000

Abstract

The Linux kernel development project was among the first attempts that make a deliberate effort to use globally connected software developers as the main source of talent and input to create an important, Open Source software. Based on the Linux project, we have built a model of Open Source knowledge creation to study how thousands of talented volunteers, who are dispersed across organizational and geographical boundaries, collaborate via the Internet to produce a knowledge-intensive product of high quality. Comparing and contrasting the Linux model with the traditional/commercial model of software development, we focus on four issues critical to software development: intellectual property licensing, incentive to contribute, coordination mechanisms, and production process. Recognizing that the applicability of the model may be constrained if business firms do not have the supporting infrastructure and work practices, we propose several areas where the model can be adapted and how the adapted models are useful to business firms.

I. INTRODUCTION

How does a distributed group of individuals, who are dispersed across space, time, and organizational boundaries, organize themselves to create a useful product? The advent of the Internet and web-based technologies has enabled specialized communities to convene, interact, and share resources extensively via electronic interfaces. One prominent example is the Open Source community, which shares the source code of the software that its members have developed collectively with anyone who wishes to download via the Internet free-of-charge.[1] The Linux operating system development project is among the first attempts to make a deliberate effort to use the globally connected software developers as its main source of talent and input to create a widely shared and used software product. As a natural experiment, the Linux project has demonstrated the feasibility of a large-scale on-line collaboration effort where developers and users can be one and the same - though over time, the proportion of "non-developer users" grows more rapidly than that of "developer users."

Linux type of phenomena are increasingly popular beyond the Open Source community as the use of the Internet increases. For example, the Open Directory Project provides the technologies for Internet users to develop an alternative directory service web site to commercial portals, such as Yahoo!. Internet users can volunteer as editors by choosing an area of expertise and collecting high quality Internet links for that category. More than 20,000 Open Directory editors have identified links to more than 1,200,000 web sites, with more than 3,000 new sites added daily.[2] Many Internet search services such as HotBot, Lycos, and Netscape Communications have recognized the Open Directory as a source of quality information.

[1] The Open Source definition is available at http://www.opensource.org/osd.html.
[2] http://dmoz.org/about.html

Our objective is to study the Linux kernel development process as a model of knowledge creation. We focus on the Linux project because we are interested in building a model that is based upon a large-scale community of organizationally as well as geographically dispersed knowledge workers using many-to-many communications technologies and sharing their intellectual properties under an Open Source license. Although there are many other Open Source projects that have successfully created products such as the Apache web server, the Perl programming language, the BIND domain name service software, and the Sendmail email transport software, the scale of those is much smaller.

Our emergent model of knowledge creation has major implications for organizational innovation in a society in which the nature of work is increasingly influenced by the Internet. We focus on four issues critical to software development: intellectual property licensing, incentives to contribute, coordination mechanisms, and production process. Each issue contains many organizational processes such as structural change and decision-making that deserve fuller theoretical attention. Due to space constraint, we present in this paper an empirical analysis of the Open Source phenomenon through the lens of knowledge creation. Our goal is to set up a foundation for more theoretical research in the future.

Our model of Open Source knowledge creation contributes not only to the development of theory, but also to the design of business strategies. For instance, our model is useful to business firms in designing their product development process and licensing strategy. Many business firms such as Sun Microsystems and Netscape Communications are experimenting with ways to benefit from innovations that occur in the Open Source community like Linux. Another key strategy implication is the design of inter-organizational relationships for the purpose of joint product innovations in industries where network effects are strong. When firms coordinate and collaborate to jointly develop new products or to achieve interoperability among their products, they form loosely-structured alliances or standards bodies by contributing their intellectual properties. An understanding of how the Linux model operates will help practitioners and researchers appreciate the conditions under which Open Source strategies are effective or alliance and partnership strategies are functional.

In section II, we review foundational literature on knowledge creation and outline the contributions of our emergent model of knowledge creation. In section III, we describe the research setting, data, and methodology. In section IV, we develop a case study and discuss in detail how the Linux development community is organized to address critical issues of intellectual property licensing, incentives, coordination, and production. In section V, we summarize our analytical model of Open Source knowledge creation and discuss its generalizability. Then in section VI, we discuss the applicability of the model beyond the Open Source community to business strategies. Finally, we conclude in section VII.

II. MODELS OF KNOWLEDGE CREATION

We apply grounded theory building (Glaser & Strauss, 1967) to develop a model of Open Source knowledge creation because the foundational literature in knowledge creation ill prepares us for a world of knowledge creation through distributed cognition and geography. This is not surprising since much of the literature was formed before the Internet became a true force in knowledge collaboration. Consider three of the most critical assumptions of those pioneers seeking to understand the process of knowledge creation.

First, there is the assumption throughout this literature that the unit of analysis and/or the locus of action takes place at the level of the firm or a set of firms. The firm is seen as the depository for knowledge and the physical locus of its creation and deployment. The firm provides the social, political, and economic context under which knowledge is constructed (Argyris, 1993; Prahalad & Hamel, 1990; Levitt & March, 1988; Winter, 1987; Teece, 1986; Nelson & Winter, 1982; Penrose, 1959). Even if the researcher focuses on the individual such as Argyris does, it is in the context of producing learning for a firm. Analogously, "communities of practice" as set out in Lave and Wenger (1991) and developed in an organizational context in Brown and Duguid (1991) are studied as the unit of analysis for understanding knowledge inside the firm. These communities are emergent groups in existing organizations, including the focal firm and its suppliers and customers. When the firm or firms are the unit of analysis, we naturally anticipate that firms will treat the knowledge created by its employees as the firm's intellectual property and seek to leverage it for competitive advantage. This involves complex calculations designed to protect proprietary knowledge under certain conditions and share it under others (Teece, 1998). Yet as the use of electronic communications makes possible collaboration by individuals outside existing organizational boundaries, the issue of how intellectual property gets treated and leveraged becomes problematic. We will show how the issue of intellectual property is handled in the Linux model of knowledge creation.

Second, the foundational scholarly literature on knowledge creation assumes face-to-face interaction among knowledge developers to facilitate trust-building over a long period of association and through sharing a common space (e.g., Nonaka & Takeuchi, 1995). This is a natural extension of using the firm as a unit of analysis because it is in the firm that physical proximity supports the development of trust through repeated interactions and shared social norms. It is no accident that Japanese firms became one of the more important sites for the building of knowledge creation theory since they have had a tradition of long term employment, which nurtures shared experiences and values. We agree that trust is a key variable in knowledge management and how knowledge works (e.g., Powell et al., 1996; Kramer & Tyler, 1996). However, as the use of electronic communications becomes more prevalent, researchers need to study the issue of trust-building in a virtual environment. We will show a mechanism that the Linux development community uses to help developers trust their intellectual properties with individuals with whom they have not had prior personal contact.

Third, the foundational literature, by relying on firms as units of analysis, typically assumes that knowledge creation takes place under conditions of authority and hierarchy with complex divisions of labor being required for producing sophisticated knowledge products. Furthermore, strong coordination measures are required to move individual knowledge to group and then organizational knowledge (Nonaka & Takeuchi, 1995). Even when relatively independent (self-organizing) product development teams are enacted, they often rely on "heavyweight product managers" (Clark & Fujimoto, 1991). Conversely, with lightweight product managers, the assumption is that the knowledge creation process exists in a wider environment of hierarchy and authority. The famous example of the creation of the breadmaking machine documented by Nonaka and Takeuchi relied on the surrounding structure of hierarchy and authority at Matsushita Electric Industrial Company just as surely as it did on the free flowing creativity of the product development group (Nonaka & Takeuchi, 1995). Yet, if one is to create complex knowledge products with increasing reliance on electronic communications for connecting organizationally as well as geographically dispersed near strangers, traditional mechanisms of hierarchical control and authoritative command appear problematic. How is a complex division of labor enacted and then coordinated in an on-line community among volunteers where no formal mechanisms of control exist? We will address the issues of coordination, social control, and worker incentives of the Linux project in this paper.

Fortunately, recent research has extended the scope of these traditional assumptions in ways that favor the creation of knowledge products through distributed cognition and geography. In our judgment, this recent research points us in the right direction, but we need to extend its reach to accommodate five simultaneous conditions: (1) A large-scale community with thousands of people; (2) Organizationally as well as geographically dispersed knowledge workers; (3) The use of a many-to-many communications technology; (4) The assignment of intellectual property in ways that promise the sharing of knowledge; and (5) The motivations and incentives of volunteer workers.

The first condition that our emergent model of knowledge creation needs to accommodate is a large-scale community with thousands of people engaged in knowledge creation. In principle, there is no reason why communities of practice should be limited to those within the same firm. The anecdotal examples of communities of practice, however, are usually of a small scale as in the case of a product development team with less than half a dozen people (Brown & Duguid, 2000: 127). The scale of *Ba,* which is the knowledge creation platform described by Nonaka and Konno (1998) and sets a broader social foundation for knowledge creation that potentially extends beyond the boundary of a firm, is also fairly limited, compared to the size of the Linux development community. The largest number of people or organizations involved in their examples was at Sharp, which worked with 600 "leading consumers" to create a customer knowledge base (Nonaka and Konno, 1998: 48). The concept of *ba* or communities of practice illustrated in these examples suggests its applications are most relevant to relatively modest sized groups. However, as productive activities, enabled by Internet connectivity, are deployed on a global scale, models of knowledge creation must be sensitive to scale.

The second condition for our emergent model of knowledge creation requires a social structure to accommodate organizationally as well as geographically dispersed knowledge workers. It challenges the assumption that face-to-face interactions are indispensable for both building trust and repairing shattered trust (Nohria & Eccles, 1992; O'Hara-Devereaux & Johansen, 1994). Recent research has shown that "swift trust" can be built over an electronic medium (Jarvenpaa & Leidner, 1999). A case in point is academic and research communities that have used emails and newsgroups as communication tools for the purpose of collaboration even before the popularization of commercial Internet technologies. Nevertheless, we argue that the Linux development community is different from academic and research communities because the collaboration occurs among mostly strangers who do not have publications and/or university affiliations to serve as proxies of their qualification, reputation, or trustworthiness. We believe our study will lead to a better understanding of how talented people (but near strangers), who are physically in disparate organizational and geographical locations, can be mobilized and organized to create a useful product like the Linux kernel.

The third condition that our emergent model of knowledge creation needs to accommodate is the existence of a technology enabling many-to-many communications. The Internet is that technology. Face-to-face interactions via the telephone/two-way radio are one-to-one communications, while television, radio, and other broadcast media are one-to-many. A popular example with which Brown and Duguid illustrated the concept of communities of practice is the community of field-technicians (Orr, 1990a, 1990b, 1987a, 1987b), who use two-way radios to share and weave stories about their experiments with a machine. We believe the communications technologies that the Linux community uses are quite different in functionality and in nature. In addition to providing many-to-many communications, the Internet also allows users to transmit text, picture/image, voice, and video, which become the objects of discussion. We will discuss the role of Internet technologies as a communication channel and as a knowledge creation platform.

The fourth condition for our emergent model of knowledge creation is the assignment of intellectual property in ways that insure the sharing of knowledge. Examples like Hewlett-Packard Laboratory's "Network of Experts" (Davenport, 1996), the use of Lotus Notes at Price Waterhouse (Orlikowski, 1993), and the use of Lotus Notes at Ernst & Young (Davenport, 1997) have demonstrated that knowledge sharing is an important strategic goal, yet fairly difficult to achieve. Our analysis will explain how a social and legal innovation such as Open Source licensing brings a creative community together and creates the conditions under which people can borrow what others have done and build upon others' work.

The fifth condition involves the motivations and incentives of volunteer workers. Linux developers are volunteers who do not receive monetary compensation from the development community itself. We raise questions about the rational actor assumption, which predicts that workers who are agents employed by a firm are inclined to shirk unless otherwise motivated. Under the rational actor assumption, problems of free-riding and the tragedy of the commons are also assumed to afflict team work. Our case study will show who the Linux developers are and why they voluntarily contribute to the development of an Open Source software.

III. RESEARCH SETTING, DATA, AND METHODOLOGY

Linux is considered to be a serious threat to Microsoft Windows NT's market dominance in operating systems. It is truly remarkable that such a system starting as a hobby in 1991 and as an Open Source software should become, by 1999, the World Wide Web's leading operating system, running 31 percent of the Web servers (versus 24 percent for Windows and 17 percent for Solaris).[3] Linux runs on computer systems of small networks, which Internet service providers and university computer labs use, as well as those of large networks, used by such organizations as Wells Fargo and the U.S. Postal Service (Mann, 1999). Recognized for its speed, reliability, and efficiency, Linux now has more than 12 million users worldwide and an estimated growth rate of 40 percent per year.[4] In this sense alone, Linux must be regarded as a success and this high utilization rate relative to other products suggests high quality.

Shortly after the emergence of electronic commerce in the mid-1990s, the demand for web serving and networking technologies by firms new and incumbent has increased tremendously. For firms whose business depends on a highly reliable operating system that functions well with network servers, Linux is a viable, lower cost choice. Combined with the Web servers, the Linux operating system enables real-time electronic commerce and information sharing. Web serving is particularly a kernel-intensive function for an operating system. In this context, Linux as a competitive operating system and as an open standard received much positive coverage by news media and attention by business firms.

An operating system is one of the five major layers of software infrastructure and it performs a number of functions, including storage management, communications, and support for task concurrency within a single host. The kernel of the operating system schedules tasks,

[3] The Internet Operating System Counter, http://www.leb.net/hzo/ioscount/index.html.
[4] http://www.linux.org/info.

which include the execution of end-user applications (e.g., web browsers, word processors, and database management systems) by allocating the computer's system resources to the programs in execution. To end-user applications, the kernel is a housekeeping unit that handles process management and scheduling, inter-process communication, device Input/Output, and memory management for the operating system. To underlying hardware, the kernel converts operating system calls into lower-level hardware programs through hardware-specific drivers.

The original creator of Linux is Linus Torvalds, who was an undergraduate student at the University of Helsinki in Finland when he started the project as a hobby. Torvalds made use of the Internet technologies and reached out to other software enthusiasts to get help developing the software program and to gather suggestions and advice on the features that Linux should contain. He sent an electronic message in July 1991 to the comp.os.minix newsgroup and asked if someone could point him to information on the posix standard definition (DiBona, Ockman, & Stone, 1999: 269-270). After a few months of work and email exchanges, he had succeeded in developing a version of the program that was reasonably useful and stable. Since he had a difficult time finding suitable and affordable operating system source code, Torvalds wanted to provide Linux to other users so they could freely modify and experiment without having to pay a costly license fee. Torvalds (1992) released the Linux kernel source code in October 1991 to the newsgroup free-of-charge and solicited other programmers to contribute computer code that they have developed to add to the Linux source code.

To date, thousands of self-motivated contributors across many countries who wanted to try their hands on cutting-edge computer science have participated in the development of Linux. Seemingly, the concern for free riding and the risk of wasting time on a lost cause would discourage computer programmers from participating. Despite these potential impediments, however, the Linux kernel was created in a little over two years and it has gone through numerous revisions since then.

Our study is grounded in archival data analyses, on-line research publications, second-hand interviews, and observations of how the technology has evolved. We study the Linux development community mainly by analyzing the artifacts that the Linux developers have produced. A key output of knowledge creation activities is the artifacts. The most important artifact, of course, is the Linux operating system source code.[5] We choose Linux 2.2.14, released in March 2000, as our main source of data because the Linux kernel development project has stabilized by version 2.2, which has been developed between 1999 and 2000. More exciting developments for the Linux operating system now take place outside the kernel (Torvalds, 1999: 111).

The Linux 2.2.14 source code has a size of 62.7 megabytes and approximately 1.9 million lines of code in 5,186 files and 266 folders. Along with the source code, a "Credits" text file and a "MAINTAINERS" text file are distributed to the users. For easy user reference, these files are located at the first level of the directory (2.2.14/Linux/) next to the folders containing modules and documentation. The Credits file is a public recognition of the people who have substantially contributed to the development of Linux kernel.[6] The file lists the names of recognized developers as well as a description of their major contributions. Similarly, the MAINTAINERS file keeps a record for each subsystem and its maintainer.

The development work takes place mainly at the Linux-kernel mailing list, which is a virtual environment where Linux developers send their contributions, discuss implementation details, and interact with other developers. Archived Linux-kernel mailing list is another

[5] Anyone has free access to the source code stored in a publicly accessible web site called The Public Linux Archive at http://www.kernel.org/pub/ for free-of-charge download

important artifact, with which we analyze patterns of development activity.[7] Using the weekly Linux-kernel email archive from years 1995 to 2000 as a key source of data, we focus on people who have sent at least one email to the Linux-kernel mailing list. We found that 14,535 people have sent at least one email to the Linux-kernel mailing list to participate in the development project between 1995 and 2000. On average, each person has sent 14 emails over five years.[8]

Using developers' email suffix in the Linux-kernel mailing list archive, Credits file, and MAINTAINERS file, we study developers' organizational affiliation and nationality. In addition, we examine the developers' demographic distribution, working patterns, and motivations by analyzing the raw data from an on-line survey (we call it "the Linux-kernel survey") that was distributed electronically by a research team at the University of Kiel in Germany to the Linux-kernel mailing list (see Appendix A). The period of data collection was between February 2000 and April 2000. Using the World Wide Web, we also study public documents related to Open Source and Linux. Some examples are OpenSource.Org,[9] Linux Frequently Asked Questions (FAQ),[10] Linux Knowledge Base search engine,[11] and The Linux Care Kernel Traffic newsletter.[12]

IV. CASE STUDY: THE LINUX KERNEL DEVELOPMENT PROJECT

Our case study focuses on four issues that are important to the understanding of the Linux kernel development process. The issues of intellectual property rights, incentives to contribute, coordination mechanisms, and production process provide answers to the following questions: If

[6] Only 28 maintainers are listed in the Credits file
[7] http://www.uwsg.indiana.edu/hypermail/linux/kernel/
[8] As of August 26, 2000, there were a total of 199,374 emails archived in the mailing list.
[9] http://www.opensource.org
[10] http://www.tux.org/lkml/
[11] http://www.linuxcare.com/help-yourself/kbsearch/simple-search.epl
[12] http://kt.linuxcare.com/

the development community, not a firm, is the unit of analysis, how can a user of a software influence the rate and direction of innovation? Also, what are the licensing mechanisms that enable developers to trust a community of strangers with their intellectual properties? Moreover, who are these developers volunteering their talents and what are their incentives to contribute? Altruism certainly is not the only motivation. Also, given the complexity of coordinating a large-scale project with thousands of developers who are dispersed organizationally and geographically while they work simultaneously on the same computer program via many-to-many communications technologies, how do these developers organize their activities to develop a successful product? What is the decision-making process and its associated organizational structure to ensure product quality? These are the questions to which we will direct our attention.

A. Intellectual Property Licensing Mechanisms

One of the most important features of the Linux development community is that users can participate as developers. There are more than 12 million Linux users world-wide and approximately 90,000 of them have registered themselves as Linux users to date.[13] We estimate 16 percent of the registered users participate as developers.[14] Linux developers carry out two important functions in the development process: (1) quality assurance and (2) innovation. For quality assurance, developers perform the tasks of bug reporting, identification, correction, and testing. For innovation, they make suggestions for new features and write patches of computer code to enhance the usefulness of Linux.

[13] http://www.linux.org/info/index.html

[14] We found 14,535 people have sent at least one email to the Linux-kernel mailing list between 1995 and 2000 to participate in the discussion of Linux development. Among approximately 90,000 people who have registered themselves as Linux users, we estimate 16% of the registered users participate as developers.

Compared to traditional software development process where end-users mostly serve as a source of bug reports and complaints, the Linux project encourages users to become problem-solvers and serve as a source of solutions and innovations. We recognize that the Linux project is not the only setting where user-driven innovations take place. Among other examples, Rosenberg's (1976) studies of the machine tool industry and von Hippel's (1988) of scientific instruments also demonstrated that sophisticated users are an important source of product innovation. Indeed, users can play a significant role of accelerating technological progress and even leading the direction of innovations. However, Linux users are remarkably active in conducting beta-testing and problem-solving, relative to other cases of user-driven innovations documented in the literature. As such, the Linux project represents an extreme case of this user-driven phenomenon, pushing the limits of the value that users add to software development.

Users can participate as developers because Open Source licensing gives software users the access to and control over the building blocks (i.e., the source code) that are used to create the software programs. Source code is the programming code that computer programmers use to write a software program. When the software is distributed in source code, programmers can modify the computer program for quality improvement, customization, and innovation. From the software developers' point of view, source code is much more flexible than machine code. However, commercial software is typically distributed in machine code, which is the only language that computers understand and execute. From a software development firm's perspective, distributing only in machine code protects the ownership of intellectual property because machine code is in a binary form and software users rarely have the capability to reverse engineer the product. Firms treat intellectual property rights as exclusion rights so as to capture monopoly rents. However, the consequence of distributing only machine code is that users cannot easily improve, customize, or extend the software.

The Linux operating system is distributed under an Open Source License. Open Source Licenses, such as the GNU General Public License (GPL) that Linux is distributed under, guarantee anyone the right to read, redistribute, modify, and use the software source code freely. Open Source licensing is a social and legal innovation designed to redefine the use of intellectual property rights. While traditional software licenses protect copyright, Open Source licenses guarantee “copyleft,” which reserves the right of users to modify the source code of the software for purposes of quality improvement, customization, and extension. Under an Open Source License, any changes or improvements to the source code is required to be made available to the public. Additionally, modifications of existing source code must be distributed under the same license as the original software.

The requirement to make any modification to the source code publicly available creates the foundation under which developers can trust others, whom they have never met, with their intellectual properties. By design, the source code and any changes to it are available to the public, so without excludability, the code has little market value. Since the pricing of the code itself is zero, a developer is left with two choices. He or she can keep the modifications private for fear of other developers' misappropriation, or exchange it for peer recognition and social status in the development community. Since other developers who modify "copylefted" source code are required to return the changes to the public, the likelihood of misappropriation is significantly reduced. In addition, the fame and glory that can result from a developer's work sometimes bring a developer employment opportunities and career advantages.

Although "copyleft" has not been challenged in the court of law, the Linux development community has developed a protocol to put the norm of "copyleft" into practice. One of the community protocols is a strong norm to properly cite authors whose work is being extended or borrowed. Such protocol shows how much this community respects the contributions and intellectual properties of its members. The extent of source code sharing and the community protocol of recognizing contributors are clearly observed at the beginning of each source code file. Reviewing the Linux kernel 2.2.14 source code, we find at the beginning of each file the name of the main author and the names of the collaborating contributors as well as their respective contributions. In some cases, there is only one developer in each file, but in others, there are many developers involved with the development of the same piece of code.

As shown in the following example, the main author, who is the copyright holder of the code, is listed right underneath the name of the file. Then a list of six collaborating contributors with their respective dated contributions follows. Notice that the source code in this file is copyrighted and one of the cited work is also copyrighted. With Open Source Licensing, innovations become cumulative as they easily build upon previous innovations without the duplication of previous work. The author of the scheduling program in the example shown here has accomplished an important task without having to rewrite a particular part of the code, namely, the spinlock portion (dated 1998-12-24).

Source Code Example: Sched.c

```
* linux/kernel/sched.c
*
*  Copyright (C) 1991, 1992  Linus Torvalds
*
*  1996-12-23            Modified by Dave Grothe to fix bugs in semaphores and
*                        make semaphores SMP safe
```

```
 * 1997-01-28      Modified by Finn Arne Gangstad to make timers scale better.
 * 1997-09-10      Updated NTP code according to technical memorandum Jan '96
 *                 "A Kernel Model for Precision Timekeeping" by Dave Mills
 * 1998-11-19      Implemented schedule_timeout() and related stuff
 *                 by Andrea Arcangeli
 * 1998-12-24      Fixed a xtime SMP race (we need the xtime_lock rw spinlock to
 *                 serialize accesses to xtime/lost_ticks).
 *                 Copyright (C) 1998  Andrea Arcangeli
 * 1998-12-28      Implemented better SMP scheduling by Ingo Molnar
 * 1999-03-10      Improved NTP compatibility by Ulrich Windl
```

B. Incentive Mechanisms

To participate in the Linux development project, a person simply subscribes to the Linux-kernel mailing list. There is no identity verification or registration required because the development community is open to anyone who has an interest. Membership is based on self-selection and each person participating in the Linux kernel development project has voluntarily chosen to be a member of the community. Among five million software programmers worldwide, fewer than 50,000 of them participate in Open Source projects (Behlendorf, 1999). In general, only one person, at most, in a hundred software programmers contributes his or her time (mainly his) to the development of Open Source software.[15]

1. A Global Workforce Across Multiple Organizational Boundaries With Different Forms of Contribution

The Linux kernel development project has attracted and utilized talented individuals who are distributed across organizational as well as geographical boundaries. With few outlets in countries outside the United States for their talents, many computer programmers in less developed economies seized the opportunity to participate in the development of Linux through the Internet. In addition to the popularity of the Internet as a communications channel, the affordability of desktop computing increases the number of people around the world who can become potential developers. There were fifteen countries represented among the 128 names listed in the Linux 1.2.0 Credits file released in March 1995. Five years later, the Credits file in the Linux 2.2.14 released in March 2000 has 196 names and there are seven more countries added (i.e., Argentina, Austria, Czech Republic, Hong Kong, Mexico, Norway, and Russia) and one country removed (i.e., Spain). Figure 1 shows that the email suffix listed in the 2.2.14 Credits file represents 21 countries. A sampling of emails listed in the Linux-kernel mailing list archive indicates that the developers are participating from at least 30 countries. The email suffix also shows a strong and steady presence of European population within the Linux community (41 percent in 1.2.0 and 42 percent in 2.2.14).

<u>Figure 1. Nationality distribution of the people listed in the 2.2.14 Credits file.</u> HERE

While users in less developed countries have actively participated and benefited from the development of a free software, collaboration takes place not only across geographical boundaries, but also across organizational ones. Using the suffix of emails in the Credits file that originate from the United States, we find that the developers have different organizational affiliations including commercial (.com), educational (.edu), as well as governmental (.gov) and other types of organizations (.net and .org) (see Figure 2). Commercial organizations (.com) is the leading category and there are many companies listed in the Fortune 500. We observe in the MAINTAINERS file the influence of commercial organizations in the Linux community as 14 of the 132 subsystems listed are supported by someone who is paid to maintain the subsystem. Nevertheless, we consider each participant as a volunteer because the Linux project does not pay any wages for anyone's contribution. Monetary compensation is provided by third parties, which

[15] Ninety-two percent of the survey respondents is male

are typically private commercial firms. The majority (64 percent) of the respondents report that they work on Linux development during their leisure time, not work time, for an average of 10 hours per week. Additionally, 60 percent of the survey-respondents have never been paid for their work on Linux, although 17 percent have been paid regular salary and another 17 percent have sometimes been paid. Universities (.edu) is another category of organization that is highly-represented in the Linux community. The Linux-kernel survey shows that 23 percent of the respondents are students. Sixty-two percent of all respondents have a full-time job and some of them are researchers at universities.

Figure 2. The distribution of organizational affiliation listed in the 2.2.14 Credits file. HERE

Linux developers not only are different in organizational affiliation and nationality, but also differ in their forms of contribution. Developers make contributions for a wide range of tasks: finding bugs, fixing bugs, testing features, writing manuals, adding capabilities, adding utilities, and porting the operating system to different computer platforms. The Linux-kernel survey shows that the majority of respondents have contributed in forms that do not require computer programming. Less than half of the respondents have actually been involved with writing computer code. Only 30 percent of the respondents have ever submitted a patch of code to the mailing list. In addition, only 45 percent of the respondents have written at least one line of code for the kernel. Further, only 45 percent of the survey respondents are involved in some subsystem project, but 47 percent are not involved in any subsystem development at all. It is common that different individuals perform different tasks because some tasks require specialists with certain type of programming skills. For example, most people can report bugs or request new features, but only some can send in patches of code to fix certain problems.

2. Market and Non-pecuniary Incentives

The incentives that motivate Linux developers to participate also vary widely. By observing email discussions, reading commentaries on Linux web sites, and analyzing interview transcripts (e.g., Ghosh, 1998), we have identified two types of incentives among these self-motivated developers. The first type is market incentives, which motivate developers who may increase their potential market value or receive monetary compensation through their involvement in the Linux project. In contrast, the second type is non-pecuniary incentives, which attract developers who may derive personal satisfaction and enjoyment from improving Linux and being a member of the Linux development community. Compared to market incentives, non-pecuniary incentives can lead individuals to contribute because participation carries its own rewards (see Deci, 1971 on intrinsic motivation). These two types of incentives are not mutually exclusive and a developer usually has multiple reasons for participation.

For market incentives, we have identified four categories of developers. The first category is the developers who are hired specifically to develop commercial distributions of Linux. The second is those who are not hired specifically to develop commercial distributions, but their day-to-day responsibilities include occasional Linux programming for their work. The third is the developers who write software applications for the Linux platform and may potentially receive monetary compensation in the form of ownership shares in start-up companies or in the form of venture capital funding for their start-up companies. Finally, the fourth is the developers whose experience with Linux increases their value in the labor market. For example, students may be motivated to work on Linux development for career concerns.

Schools and universities use Linux as an educational tool and students may likely view learning Linux as an investment in their general human capital that is transferable to work settings later.

On the contrary, non-pecuniary incentives motivate individuals who have fun programming and enjoy collaborating with other Open Source developers to create a better tool. The first type of non-pecuniary incentives is solution sharing and joint problem solving, which is associated with an anticipated reciprocity. Developers have a problem to solve (or an "itch to scratch") and the problem turns out to be a common frustration for many other programmers. By sharing solutions, the developers receive feedback and alternative solutions to the problem initially posed. The second type is the joy of craftsmanship, which is satisfaction derived from expressing a developer's talents and abilities to himself (see Veblen, 1914, on instinct of workmanship). The third category is the attention, cooperation, and recognition from one’s peers. This incentive originates from the need for social recognition and peer approval. Lastly, the fourth category is group identity. Through an electronic medium, Linux developers share experiences, form friendships, and develop a strong attachment to the Linux community. We consider the first three categories as expressions of the ego or personal needs, but the last category, group identity, derives from a sense of altruism. A sense of satisfaction is derived from contributing to a greater good (see Piliavin & Charng, 1990, for a review on altruism). Again, these are not mutually exclusive categories.

The Linux-kernel survey asked the survey respondents to rank several statements revealing their motivations and reasons for participating in the development project. We tabulated the percentage of the survey respondents that consider the statement to be true and agree with it strongly. We also cross-tabulated the responses with the developer's profession and compensation (see Table 1). We consider statements such as improving programming skills, facilitating daily work, and gaining career advantages as indicative of market incentives. On the other hand, we regard statements like having fun programming, believing that information should be free, interacting with other software developers, and dismissing the importance of monetary compensation as indications of non-pecuniary incentives.

There are statements that reflect both market incentives and non-pecuniary incentives. Improved kernel quality can better facilitate a user's daily work and at the same time serve as a demonstration of a craftsman's skill. Gaining reputation can increase a developer's value in the labor market and at the same time be an achievement among peers. This category is particularly interesting because individuals may initially participate for one reason but later reaps the benefits of others as well. More importantly, in the case of improving kernel quality, individuals may be motivated by incentives important to themselves, but the effort that they contribute in aggregate leads to a better tool not only for themselves, but also for others.

<u>Table 1. Motivations and Beliefs of Linux Developers (as of year 2000)</u> HERE

Overall, "Having fun programming" (a non-pecuniary incentive) and "Improving programming skills" (a market incentive) received the highest level of agreement across all respondents. For students and developers who have never been paid for their contribution to Linux development, "Improving programming skill" (a market incentive) received the highest level of agreement. Among developers that are paid sometimes, "Having fun programming" (a non-pecuniary incentive) is the most popular statement. Respondents, who are paid regular salary, rank "Using better software to facilitate their daily work" (a market incentive) to be the most important incentive.

Several researchers have tried to characterize why programmers volunteer in Open Source projects. For example, Lerner and Tirole (2000) found evidence in the Apache web server project that the main incentives are user benefits and reputational benefits. Raymond (1999) also noted reputational benefits as the primary incentive in the Linux development community. However, our analysis of the Linux-kernel survey data shows, on average, only 22 percent of the respondents agree that gaining a reputation as an experienced programmer in the community is very important to them. Even among developers who work on Linux for a living, only 35 percent consider reputational benefits to be very important. Our data show that reputational benefits are of second order concern while the joy of craftsmanship and learning benefits are of first order concern to Linux developers.

We speculate that the discrepancy between others' findings and ours is a difference in source of opinion. As we show in the next section that the Linux community has a two-tier structure and that the core of the two-tier structure consists of a project leader and hundreds of maintainers. We believe individuals in the core are more likely than individuals in the periphery to receive and care for reputational benefits. One piece of supporting evidence for this view is that maintainers (33 percent of the survey respondents) rank gaining a reputation as very important, but across all respondents, only 22 percent consider reputation as very important. Nevertheless, even Torvalds spoke of reputational benefits as a second-order incentive during an interview.

> *"Originally Linux was just something I had done, and making it available was mostly a "look at what I've done - isn't this neat?" kind of thing. Hoping it would be useful to somebody, but certainly there is some element of "showing off" in there to.*
>
> *"The 'fame and reputation' part came later, and never was much of a motivator... A large motivator these days (and this started to happen pretty quickly after making it available) was just that people started using it and it feels good to have done something that other people enjoy using.*

"I've never personally been in the position that I felt I had to program for money - programming may be my job, but long before it was my job it was my pleasure. So the concept of making money or even just fame off software was really fairly secondary to the fact that I wanted to program anyway.
(quoted from Ghosh, 1998)

C. Coordination Mechanisms

When there are only few people developing a system, the project team can easily modify and improve the code without interfering one another. As the operating system becomes more complex and there are more active developers, increasing organizational size and system complexity cause problems of coordination. The Linux community has in place an explicit coordination mechanism as well as an implicit one to set the boundaries where certain decisions are made centrally and others locally. The explicit mechanism is the modular structure of the code and the implicit one is a two-tier structure for division of labor. We will discuss in this section how the Linux development community solves problems of coordination through modularity and division of labor.

1. Modularity and The Evolution of Code Structure

An annual count of the emails in the Linux-kernel mailing list shows that the size of the development community has grown four times between 1995 to 2000. The larger the size of the community, the more difficult it is to coordinate and understand all the possible interactions among software components developed by different parts of the community. The positive correlation between organizational size and the delay in time to market is typically known as the "Brooks' Law." In The Mythical Man-Month, Fred Brooks (1995) argued that adding more programmers to a software project at a late stage of the development process further delays the project. He reasoned that the complexity and communication cost of a project increase with the square of the number of developers, but the amount of work done only rises linearly.

Concerned about problems of coordination, Andrew Tanenbaum, a computer science professor and a well-respected researcher, addressed, in an email to Torvalds in 1992, the importance of control in achieving a successful operating system project. Tanenbaum forcefully argued that it is critical to have someone maintain tight control of the code so that its complexity does not explode and the core of the system does not fragment into different directions:

> *... The problem is co-ordinating things. Projects like GNU, MINIX, or LINUX only hold together if one person is in charge. During the 1970s, when structured programming was introduced, Harlan Mills pointed out that the programming team should be organized like a surgical team – one surgeon and his or her assistants, not like a hog butchering team – give everybody an axe and let them chop away. Anyone who says you can have a lot of widely dispersed people hack away on a complicated piece of code and avoid total anarchy has never managed a software project*
> (quoted from DiBona, Ockmanm & Stone, 1999: 247).

To resolve problems of coordination, Torvalds decided to add loadable kernel modules in Linux 2.0.0 released in 1996 to set the boundaries within which the developers of each module have full control over implementation and design details. When a large number of people jointly develop a computer program, minor modifications in one part of the program may require significant changes and major rework in other parts of the program. “Without modularity, I would have to check every file that changed, which would be a lot, to make sure nothing was changed that would effect anything else. With modularity, when someone sends me patches to do a new filesystem and I don’t necessarily trust the patches *per se*, I can still trust the fact that if nobody’s using this filesystem, it’s not going to impact anything else" (Torvalds, 1999: 108, emphasis is original). Therefore, modularity grants developers the freedom to work on different parts of the system simultaneously without the risk of interfering with one another's progress.

With modularity, coordination is achieved not only across different parts of the system, but also over time for continuation and consistency. In a virtual setting, Linux developers can easily join and exit the community as they please. The difference between the number of active developers per year (e.g., 6,000 people in 2000) and the total number of participants across five years (14,500 people) indicates the possibility that the Linux kernel development community has had at least two complete turnovers. If the original developer of a module leaves the community, someone else who joins the community later can continue the work within the module more easily. Modularity makes tasks more explicit and clearly defined.

2. Division of Labor and Distribution of tasks

Based on developers' tasks, roles, and levels of involvement, we have identified three categories of participants in the Linux kernel development community. We observe an emergent division of labor in the Linux community, which has a project leader, several hundreds of maintainers, and several thousands of developers, including "the development team" and "the bug reporting team" (see Table 2). Each category of developers has specific tasks and roles. Roles emerge in the process of performing tasks, as opposed to being planned or someone being assigned to carry out a task.

Table 2. Emergent Division of Labor HERE

Torvalds is the founder and the default project leader. Because of the complexity of the project, he is assisted by a group of maintainers who are responsible for various subsystems. The MAINTAINERS file lists 121 maintainers in charge of 132 subsystems. Some subsystems have co-maintainers and some maintainer watches over more than one subsystem. The third category is the developers who organize themselves into two teams that we call the development team and the bug reporting team.

We define the development team as the developers who have sent at least one email with the word "PATCH" in the subject heading between 1995 and 2000 either to send a patch of code or to discuss a patch. Their tasks include creating patches, adding features, and fixing bugs. There are 2,605 people in the development team over the five year period. Multiplying the average number of developers working on a subsytem, which is 10 according to the Linux-kernel survey, with the number of subsystems (there are 132 subsystems), we find that the size of the development team working on subsystems is 1,320 people. The order of magnitude is consistent with our count of "patch" senders.

The other team of developers is "the bug reporting team" and we define it as the developers who have sent at least one email with the word "OOPS" in the subject heading either to report a bug or to fix a bug. [16] Their tasks include identifying bugs, characterizing bugs, and eliminating bugs. We have found 1,562 people in the bug reporting team between 1995 and 2000. We have also found some overlap between the development team and the bug reporting team. Forty-nine percent of the bug reporting team have sent an email with the word "PATCH" in the subject heading, while 29 percent of the development team have sent an email with "OOPS" in the subject heading.

In our opinion, the Linux development community has a two-tier structure as an implicit coordination mechanism to reduce the problems of coordination. The two-tier structure consists of a small core with the project leader and hundreds of maintainers and a large periphery with

[16] "OOPS" differs very slightly from a bug. A bug exists when something (in the kernel, presumably) doesn't behave the way it should, either with a driver or in some kernel algorithm. When the kernel detects that something has gone wrong, it generates a oops message. So, oops is a specific case of a bug. A person can find a bug, but the kernel may not generate an oops message.

thousands of developers. This structure, with a core and a periphery, is an emergent organizational form designed for the production of Open Source software. We will discuss the production mechanisms in the next section.

Although the Linux kernel development community has thousands of developers writing code and fixing bugs, the distribution of Linux developers is uneven across all nine modules listed under 2.2.14/Linux/. The device drivers module, which is the largest one, attracts 72 percent of the maintainers, who are spread across 55 percent of all the subsystems creating more than 58 percent of total kernel size in megabytes and 63 percent of total lines of code. The second largest module is only a quarter of its size. Given the very large number of manufacturers and hardware models available to computer users, it is reasonable that most of the development activities in the Linux development community focus on creating an interface to each peripheral device that some developer is interested in attaching to the computer. Compared to other modules, device drivers are relatively less complicated to write, but they tend to be difficult to debug.[17]

D. Production Mechanisms

As just mentioned, the Linux kernel development community has a two-tier structure: a core and a periphery. As increasing organizational size and system complexity lead to problems of coordination, one solution is highly-centralized decision making (see Hage, 1965) in order to maintain efficiency. In a two-tier structure, decisions on selecting code for the next official release are made centrally in the core, while those on reviewing code for implementation and

[17] Writing a device driver, http://wihok.8m.com/linux/rhl53.htm. More challenging and complex modules are kernel, memory management, and inter-process communications.

design details are decentralized in the periphery. The core relies on the periphery to generate patches of computer code, bug reports, and comments/suggestions as well as evaluations and improvements of each others' work-in-progress. We argue that this emergent organizational form enhances the evolutionary nature of Open Source software development because incremental changes to the source code as well as evaluations of the changes are produced in the periphery for the core to choose based on certain selection criteria.

1. Code submissions and bug reports

The Linux development community has documented in the Linux-kernel mailing list Frequently Asked Questions (FAQ) on the web and in the MAINTAINERS file many suggestions on how to submit patches and report bugs. These guidelines cover code testing, code submission, problem documentation, coding style, credit recognition, and intellectual property rights. The consequence of rule violation is reduced likelihood that the violator's contribution will be taken seriously or even noticed.

> *Please try to follow the guidelines below. This will make things easier on the maintainers.*
>
> *1. Always _test_ your changes, however small, on at least 4 or 5 people, preferably many more.*
>
> *2. Try to release a few ALPHA test versions to the net. Announce them onto the kernel channel and await results. This is especially important for device drivers…*
>
> *3. Make sure your changes compile correctly in multiple configurations. In particular[,] check that changes work both as a module and [as a part] built into the kernel.*
>
> *4. When you are happy with a change[,] make it generally available for testing and await feedback.*
>
> *5. Make a patch available to the relevant maintainer in the list… One job the maintainers (and especially Linus) do is to keep things looking the same… See Documentation/CodingStyle for guidance here.*

PLEASE try to include any credit lines you want added with the patch…
PLEASE document known bugs. If it doesn't work for everything or does something very odd once a month, document it.

6. Make sure you have the right to send any changes you make. If you do changes at work, you may find your employer owns the patch[,] not you.

7. Happy hacking.
(quoted from the 2.2.14 MAINTAINERS file)

There are also guidelines for bug reporting. For a bug to be removed from the source code, it is obvious that someone has to identify its existence. When reporting a bug, it is recommended to describe the problem and the conditions under which the bug can be observed. Such guideline helps other developers who contribute in different forms to coordinate their effort. For instance, reviewing the bug report submitted, some developers can characterize the bug. With the bug characterization, other developers can fix the bug by writing new code. Still others, who receive the bug fixes through the mailing list, can test the work-in-progress to verify that the problem has been corrected in the new code and that no new bugs have been introduced in the process. Eventually, the work-in-progress is integrated and released to the public either as a patch or as a part of the next version.

What follows is a suggested procedure for reporting Linux bug…

If the failure includes an "OOPS:" type message in your log or on screen[,] please read "Documentation/oops-tracing.txt" before posting your bug report… Send the output [to] the maintainer of the kernel area that seems to be involved with the problem… If it occurs [repeatedly,] try and describe how to recreate it. That is worth even more than the oops itself… If you are totally stumped as to whom to send the report, send it to [the Linux mailing list]…

This is a suggested format for a bug report sent to the Linux kernel mailing list. Having a standardized bug report form makes it easier for you not to overlook things, and easier for the developers to find the pieces of information they're really interested in…
(quoted from a file called "Reporting-bugs," which is distributed with the source code.)

2. Peer Review and Code Selection

In our judgment, code selection requires a delicate balance between encouraging innovative contributions and keeping the community unified. The Linux kernel relies upon Open Source licensing and the social nature of peer review for code selection to help the development community remain undivided. When developers independently create different versions of an important module, the system may split, or fork, into incompatible variations. Forked into many incompatible proprietary versions, Unix is an example where different development teams independently make changes that are not available to other development teams.

While the decision-making power within each module is decentralized, the control over kernel official release is centralized. The project leader and maintainers select conservatively among submitted patches and bug fixes to incorporate into official releases. Only 23 percent of the Linux-kernel survey respondents report ever having their submitted patches selected to be a part of an official kernel release. As documented in the FAQ, the code has to (1) appear "obviously correct" to Linus Torvalds, (2) receive the maintainer's approval, and/or (3) has been well tested by other developers in order to be included in the official release.[18] In principle, Torvalds, the project leader, has the final authority to decide which code becomes included in the kernel for official release. Although some popular myth equates his centralized decision-making with dictatorial control, Torvalds, in practice, often consults with maintainers on key decisions, particularly on issues that concern subsystems in which maintainers have invested time and for which they have taken responsibility.

[18] Listed in the Linux Frequently Asked Questions (FAQ), under "How do I get my patch into the kernel?"

Typically, the project leader and the maintainers select code for inclusion based on technical merits.[19] When technical merits are not immediately evident, the better the code is documented, the more relevant it is to current official release, and the more persistent the contributor is, the more likely the code will be included in the official release.[20] As suggested in the MAINTAINERS file, "*Always _test_ your changes, however small, on at least 4 or 5 people, preferably many more.*" The likelihood of code inclusion is primarily a function of the number of developers that have reviewed the code and the reputation of the reviewer(s). As such, the decision-making process is highly social in nature and peer reviews are a key input to code selection.

As an evaluation mechanism, peer review functions as an important step for quality assurance in the development process. By having many different peer developers review the posted code, the original developer(s), who may overlook certain glitches or lack the experience to solve the problems, gain extra sets of eyes to catch mistakes, identify problems, and improve quality. Peer review leverages the diverse background and work experience of many developers, who in aggregate have a broader set of tools to perform bug identification, characterization, and elimination.

Peer review also shifts quality control from a downstream detection process to an upstream prevention process by testing code at the level of initial and small changes when bugs can be more easily observed. In one of the email discussions on Linux-kernel mailing list in

[19] As such, it is claimed that there is no single company directing the development path of the Linux kernel, "Demystifying Open Source: How Open Source Software Development Works," An Industry Briefing Paper by Linuxcare, Inc. October, 1999).
[20] "The Development Process Criticized," Kernel Traffic, 14 Sep 1999 - 16 Sep 1999 (53 posts): Accountability; http://kt.linuxcare.com/kernel-traffic/kt19990927_36.epl#10

August 1999, Torvalds elaborated the merit of peer review and emphasized the importance of submitting small patches with incremental changes:[21]

> … *Common mistake: peer review does NOT mean that the code should be looked at by the same people who write it. Peer review is _meaningless_ under those circumstances. The whole point of getting peer review is to find _different_ people who have a different background to look at your code… The point of open development is that people see what's going on… You want to have random people just see small updates - because they will often catch silly mistakes.*
> *(quoted from an email sent by Linus Torvalds to the Linux-kernel mailing list in August 1999)*

One way to encourage peer review is to increase product release frequency and shorten product cycle. The sooner the feedback is incorporated, the more developers are encouraged to contribute and engaged in the development activities. Compared to traditional/commercial software, Linux is a continuously evolving product of a much higher update frequency. Most commercial software companies release their products and/or follow-up upgrades only every few years. Although commercial firms use daily build to update progress, the released information is only circulated in the firm internally. Since the first release of Linux, there has been on average one new version of the system released every week. Tables 3 and 4 show a chronology of the official code release frequency for the stable version and the experimental version of the Linux kernel source code, respectively.

Table 3. A Chronology of Stable Releases HERE

Table 4. A Chronology of Development Releases HERE

[21] "Code Freeze; ISDN Perennial Lateness," Kernel Traffic, Aug. 3, 1999 - Aug. 10, 1999 (44 posts): Re: no driver change for 2.4?; http://kt.linuxcare.com/kernel-traffic/kt19990819_31.epl#9

V. THE LINUX KERNEL DEVELOPMENT AS A MODEL OF KNOWLEDGE CREATION

The Linux model has four key mechanisms to address critical issues of intellectual property licensing, incentives, coordination, and production in knowledge creation. Open Source licensing creates a social environment where developers can collaborate to improve, extend, and customize existing software. Distributed across organizational and geographical boundaries, developers use the Internet technologies to communicate, share, and enhance each other's work-in-progress. Developers, who have multiple incentives and contribute in different forms, choose different parts of the computer program to work on according to their interests and skills. Serving the function of quality assurance and as a source of innovation, developers are coordinated in an emergent organizational form where peer reviews and incremental innovations are generated for the production of a knowledge-intensive product with high quality and useful features.

What sets Linux apart from other operating systems in the level of quality and performance is not anything inherent in the original architecture. As Raymond (1999) argued, the quality and performance of the product result from its open and evolutionary development process. Most software products, both commercial and non-commercial, have had always been produced in a "cathedral," by isolated teams of programmers who worked on the code until releasing a final, finished version. The project leader makes the decisions on the design as well as the details of implementation. Linux, on the other hand, was assembled in a "bazaar," by a group of organizationally and geographically distributed programmers, who have a wide variety of interests, needs, and abilities. Nevertheless, our argument is not that the practices of modular flexibility, parallel development, and peer review are unique to the Linux development model.

In fact, the practices just mentioned are commonly used in commercial software firms. The differentiation is the nature of Open Source, which significantly increases the possible scale and the number of developers involved in these practices.

This difference is not trivial. The essence of Open Source knowledge creation is that it enables massive knowledge sharing by significantly increasing the scale and the number of developers involved in development. This ultimately differentiates the Linux model from the traditional/commercial model of software development. The practice of knowledge sharing enhances the benefits of peer review and leads to a product of higher quality. The openness of the development process allow participants, who have different backgrounds, to review incremental changes to the source code. Moreover, sharing the source code also helps increase the number of users who are eager to switch to an open standard because an open standard frees users from lock-in to a proprietary platform. As the number of users increases, network externality makes the software more valuable; hence, the software attracts more users and increases the potential size of the development community.

However, the Open Source model of knowledge creation may have limited generalizablity to business firms if the model is not carefully emulated. For instance, the size of the development community is one variable that firms will not be able to imitate easily. The probability of finding someone solving a similar problem is lower within a firm's boundary than outside a firm. We argue that a firm needs to identify the effective size of its project team. When a project team reaches its effective size for optimal efficiency, product quality and speed of innovation may level off or even decline. Therefore, for cases where the effective size is larger than the total number of volunteers, or cases where there are considerable costs associated with adding more people to the project, the generalizability of the model in a business firm is restricted.

Another limitation is the functionality of the Open Source model. As a natural experiment, the model is only observed in cases where there is a massive army, organized for mundane, labor-intensive tasks such as debugging, reviewing code, fixing code, and adding device drivers. So far, it has not been observed in cases where there is a disruptive innovation or technological breakthrough. Bill Joy, who has created and distributed an open-source version of Unix two decades ago at the University of California at Berkeley, argues that the Internet has made collaboration easier, but real innovation remains the work of a few. "The truth is, great software comes from great programmers, not from a large number of people slaving away," said Joy (Lohr, 2000). Nevertheless, more than 75 percent of the time and cost of a software project is typically consumed by the mundane work, for which the Linux model has demonstrated its power to increase the speed of development as well as the quality of product.

VI. BUSINESS IMPLICATIONS AND OPEN SOURCE STRATEGIES

We consider the Linux model to be a pure form of Open Source knowledge creation among strangers who leverage the Internet technologies to jointly develop a useful and widely shared product. Although we recognize the limits of the pure form, we argue that the pure form can be adapted and its adapted form is useful to business firms.

A. An Open Source Model for Knowledge Sharing Inside the Firm

First, the Open Source model can be adapted for knowledge production in other knowledge-intensive industries. For-profit examples include firms in the high-tech industry and the consulting industry that rely for their competitiveness upon knowledge creation and regeneration. Following the spirit of Open Source licensing, business firms may adapt our model and promote knowledge sharing within the boundary of the firm without losing their intellectual property rights to the public.

However, current work practice in commercial software companies constrains the effectiveness of the adapted model. For example, Valloppillil (1998), a manager at Microsoft, has identified the difficulties to implement the code sharing practice at his company. He argued that because each software development group is largely autonomous, software routines developed by one group are not shared with others. In some instances, the groups may defend their boundary by strategically not documenting a large number of program features. As such, commercial software firms need to undergo major restructuring of work practices in order to reap the benefits of an Open Source model inside the firm.

B. An Open Tools Strategy for Software Licensing

We also argue that the pure form can be adapted to improve the design of business strategies. The first area of adaptation applies to a software firm's licensing strategy by combining open and closed models of software development. A commercial software firm can adapt the Open Source licensing aspect of the model and emulate some of the conditions that the model employs to design an "Open Tools" strategy. An "Open Tools" strategy provides source code access to individual developers, independent software vendors, and application providers to ensure that the software developed will remain open and interoperable with other technologies.

Sun Microsystems, for example, is in the process of setting up an infrastructure to support an "Open Tools" strategy. The company has announced four different source code licenses and among those, one is Open Source.[22] It is important to recognize that without an infrastructure,

[22] All of the licenses allow free-of-charge access, but the firm treats its four source code licenses as reserving different degrees of stewardship. The Free Solaris Source License, which is the most proprietary, maintains Sun as

an Open Source license by itself does not guarantee the success of an "Open Tools" strategy. The issues Sun Microsystems is focusing on include tools, frequency of source postings, mailing list for discussion, and the decision-making process on code inclusion to the official release of the next version.

In contrast, without proper infrastructure and work practices in place, the Netscape Communications Open Source project unfortunately was delayed by strategy changes, internal controversies, and, ultimately, defections after Netscape agreed to be acquired by America Online in November 1998.[23] As we have shown in the case study, modularity is a mechanism to reduce problems of coordination in large-scale projects. However, the Netscape program's source code was a large patch that was not designed in modules. In addition, the practice of knowledge sharing was limited. At first, the Netscape programmers were reluctant to post their comments in the online area accessible to outsiders, preferring to post them instead on in-house lists for Netscape engineers. As such, Open Source licensing is a necessary condition, but not a sufficient condition in designing an Open Tools strategy.

C. An Open Document Strategy for Joint Development and Standards Setting

The second strategy area where the Linux model can be adapted is to apply the model to a firm's design of inter-organizational relationships so the firm can effectively manage a group of loosely structured people or entities for the purpose of collaborative innovation. In practice, firms create business-to-business alliances or standards organizations to coordinate

---

the sole steward of Solaris. The Sun Community Source License allows the community of developers who have agreed to the license to share information and code with each other without Sun in the middle. The Sun Industry Standards Source License, which is under review by Open Source Initiative, was written following the open source definition to use in cases where a Standards Body is acting as the steward for a technology. The Mozilla Public License, which is compliant with the Open Source Definition, distributes Sun source code for public stewardship. See Sun Microsystem's press release on March 13, 2000

[23] Netscape Communications, a commercial software firm, launched in January 1998 an Open Source project—Mozilla.org, by releasing the source code of its Mozilla Internet browsing software. Its goal is to use Mozilla code as the basis of its Netscape 6 product.

interoperability among their products. As a part of the collaborative effort, firms often contribute their intellectual capital to standards organizations. However, alliance contracts are not self-enforcing. In this respect, alliance contracts are similar to Open Source licenses. The objective is to leverage the alliance to share one another's resources and create new resources, without jeopardizing each member's own interests. The challenge is to organize among firms where there isn't overarching hierarchical authority in place.

Like the Linux developers, firms can take advantage of the Internet technologies, which provide standardized communications protocols and connect different computing platforms across the globe, to promote collaboration. Further, firms can emulate the Linux model to coordinate and control product development and standards setting processes. We observe that among standards organizations, the degree of access to documents and participation rights varies widely. Many standards organizations restrict access to documents and meetings to members only. However, organizations like IETF, which defines standards for the Internet, makes all of the documents as well as all of its mailing lists and meetings openly available (Bradner, 1999). In this sense, the "open document" strategy adopted by IETF is analogous to Open Source licensing used by the Linux community. An open document strategy may serve to be an innovative strategy for standards competition.

VII. CONCLUDING REMARKS

This paper attempts to apply inductive theory building (e.g., Eisenhardt, 1989) to develop a model of Open Source knowledge creation. The case study on the Linux kernel development process presented in this paper shows how a large number of volunteers distributed across organizational and geographical boundaries has succeeded in collaborating in the development of a complex, high-quality, and knowledge-intensive product via many-to-many communications technologies. Contrary to popular myth that large-scale self-organizing projects tend to be anarchical, our data and analyses show that there are intellectual property licensing mechanisms, incentive mechanisms, coordination mechanisms, and production mechanisms in place in the Linux development community to ensure product quality and speedy development. Although we recognize the limitations of the Open Source model, we argue that the model we presented is a pure form that can be adapted for knowledge production outside the Open Source community as well as for the design of business strategies.

APPENDIX A: "The Linux-kernel survey"

Three researchers at the University of Kiel, Germany conducted a project in year 2000 to study why so many skilled software experts are willing to contribute their time and expertise for free in Open Source software development. Guido Hertel, an assistant professor at the University, joined by a physicist, Sven Niedner, and a student of psychology, Stefanie Hermann, created a web site, http://www.psychologie.uni-kiel.de/linux-study/, and administered a questionnaire on the Internet to survey participants on the Linux-kernel mailing list. The questionnaire was posted to the mailing list on February 15, 2000 and 142 responses were returned by April 12, 2000. After collecting questionnaire responses, they provided the raw data on the their web site for anyone to download.

We estimate the response rate of the Linux-kernel survey to be 2.4 percent.[24] Although the survey response rate is quite low, we believe the possible bias of over-sampling maintainers and active developers is not significant. The proportion of the survey respondents who claim to be maintainers and active developers (22.7 percent) is consistent with our estimate of the proportion of mailing list participant (18.7 percent) who are maintainers and active developers.[25]

[24] Based on our estimated 6,000 email senders to the Linux-kernel mailing list in year 2000. The response rate would be even lower if we account for everyone who subscribes to the Linux-kernel mailing list but never sends any email to the mailing list.
[25] In Table 2, we show the estimated size of maintainers and the development team combined is 2,726 and that is 18.7% of the 14,535 people that have sent at least one email to the Linux-kernel mailing list during 1995 and 2000.

REFERENCES

Argyris, C. (1993), *Knowledge for Action: A Guide to Overcoming Barriers to Organizational Change*, San Francisco, CA: Jossey-Bass.

Behlendorf, B. (1999), "Open Source as a Business Strategy," *Open Sources: Voices from the Open Source Revolution*, DiBona, Ockman, & Stone (eds.), Sebastopol, CA: O'Reilly & Associates.

Bradner, S. (1999), "The Internet Engineering Task Force," *Open Sources: Voices from the Open Source Revolution*, DiBona, Ockman, & Stone (eds.), Sebastopol, CA: O'Reilly & Associates, Inc.

Brooks, F.P. (1995), *The Mythical Man-Month: Essays on Software Engineering*, Reading, MA: Addison-Sesley

Brown, J.S., and P. Duguid
(1991), "Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning, and Innovation," *Organizational Science*, 2(1): 40-57.
(2000), *The Social Life of Information*, Boston, MA: Harvard Business School Press.

Clark, K.B. and T. Fujimoto (1991), *Product Development Performance : Strategy, Organization and Management in the World Auto Industry*, Boston, MA: Harvard Business School Press.

Davenport, T. (1996), "Knowledge Management at Hewlett-Packard, Early 1996," *Knowledge Management Case Study*, http://www.bus.utexas.edu/kman/hpcase.htm

Davenport, T. (1997), "Knowledge Management at Ernst & Young," *Knowledge Management Case Study*, http://www.bus.utexas.edu/kman/e_y.htr

Deci, E. (1971), "The Effects of Externally Medicated Rewards on Intrinsic Motivation," *Journal of Personality & Social Psychology*, 18, pp 105-115

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

Eisenhardt, K.M. (1989), "Building Theories from Case Study Research," *Academy of Management Review*, 14, 4, pp. 532-550.

Ghosh, R. (1998) "FM Interview with Linus Torvalds: What Motivates Free Software Developers?" *First Monday Peer-reviewed Journal On the Internet*, http://www.firstmonday.dk/issues/issue3_3/torvalds/index.html.

Glaser, B.G., and A.L. Strauss (1967), *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Hawthorne, NY: Aldine Publishing company.

Hage, J. (1965), "An Axiomatic Theory of Organizations," *Administrative Science Quarterly*, 10, pp. 289-320.

Jarvenpaa, S. and D.E. Leidner (1999), "Communication and Trust in Global Virtual Teams," *Organization Science*, 10, 6, pp. 791-815.

Prahalad, C. K. and G. Hamel (1990), "The Core Competencies of the Corporation," *Harvard Business Review*, 68, pp. 79-91.

Kramer, R.M. and T.R. Tyler (eds.), (1996), *Trust in Organizations: Frontiers of Theory and Research*, Thousand Oaks, CA: Sage Publications.

Lave, J.C. and E. Wenger (1991), Situated Learning: Legitimate Peripheral Participation, New York, NY: Cambridge University Press.

Lerner, J. and J. Tirole (2000), "The Simple Economics of Open Source," *NBER*, w7600.

LeVitt, B. and J. March (1988), "Organizational Learning," *Annual Review of Sociology*, 14, pp. 319-340

Lohr, S. (2000), “Code Name: Mainstream. Can 'Open Source' Bridge the Software Gap?” *New York Times*, Technology section, August 28.

Mann, C.C. (1999), Programs to the people, *Technology Review*, Jan/Feb, 102, 1, pp. 36-42.

Nelson, R.R. and S.G. Winter (1982), *An Evolutionary Theory of Economic Change*, Cambridge, MA: Belknap Press of Harvard University Press.

Nohria, N. and R.G. Eccles (1992), "Face-to-face: Making network organizations work," N. Nohria and R.G. Eccles (eds.), *Network and Organizations*, Boston, MA: Harvard Business School Press, pp. 288-308.

Nonaka, I. and N. Konno (1998), "The Concept of "Ba": Building a Foundation for Knowledge Creation," California Management Review, 40, 3, pp. 40-54.

Nonaka, I. and H. Takeuchi (1995), *The Knowledge Creating Company*, New York, NY: Oxford University Press.

O'Hara-Devereaux, M. and R. Johansen (1994), *Global Work: Briding Distance, Culture, and Time*, San Francisco, CA: Jossey_Bass.

Orlikowski, W.J. (1993), "Learning from Notes: Organizational Issues in Groupware Implementation," *Information Society*, 9, 3, July-September, pp. 237-250.

Orr, J.

(1990a), "Talking about Machines: An Ethnography of a Modern Job," Ph.D. Thesis, Cornell University.
(1990b), "Sharing Knowledge, Celebrating Identity: War Stories and Community Memory in a Service Culture," D.S. Middleton and D. Edwards (eds.), *Collective Remembering: Memory in Soceity*, Berverley Hills, CA: Sage Publications.
(1987a), "Narratives ad Work: Story Telling as Cooperative Diagnostic Activity," *Field Service Manager*, June, pp. 47-60.
(1987b), *Talking about Machines: Social Aspects of Expertise*, Report for the Intelligent Systems Laboratory, Xerox Palo Alto Research Center, Palo Alto, CA.

Penrose, E.T. (1959), *The Theory of the Growth of the Firm*, Oxford, England: Blackwell.

Piliavin, J.A. and H. Charng (1990), "Altruism: A Review of Recent Theory and Research," *Annual Review of Sociology*, 16, pp. 27-65.

Powell, W.W., K.W. Koput, and L. Smith-Doerr (1996), "Interorganizational Collaboration and the Locus of Innovation: Networks of Learning in Biotechnology," *Administrative Science Quarterly*, 41, pp. 116-145.

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

Rosenberg, N. (1976), *Perspectives on Technology*, Cambridge, MA: Cambridge University Press.

Teece, D.
(1998), "Capturing Value from Knowledge Assets: The New Economy, Markets for Know-How, and Intangible Assets, California Management Review, 40, 3, pp. 55-79
(1986), "Profiting from Technological Innovation: Implications for Integration, Collaboration Licensing and Public Policy," *Research Policy*, 15 (December), pp. 285-305.

Torvalds, L.
(1999), "The Linux Edge," *Open Sources: Voices from the Open Source Revolution*, DiBona, Ockman, & Stone (eds.), Sebastopol, CA: O'Reilly & Associates.
(1992), Emails sent by Torvalds in 1992 to the comp.os.minix newsgroup, archived by Linux International at http://www.li.org/linuxhistory.php

Valloppillil, V. (1998), "Open Source Software: A (New?) Development Methodology," [also referred to as "The Halloween Document"], Unpublished working paper, Microsoft Corporation. http://www.opensource.org/halloween/halloween1.html.

Veblen, T. (1914), *The instinct of workmanship : and the state of industrial arts*, New York, NY: Viking.

von Hippel, E. (1988), *The Sources of Invention*, New York, NY: Oxford University Press.

Winter, S.G. (1987), "Knowledge and Competence as Strategic Assets," David Teece (ed.), *The Competitive Challenge*, Harper and Row, pp. 159-184.

TABLES

Table 1. Motivations and Beliefs of Linux Developers (as of year 2000)

Percentage of Survey Respondents "Agree Strongly"	All Respondents (154 cases)	Student (36 cases)	Never paid (93 cases)	Paid sometimes (26 cases)	Paid Regular Salary (26 cases)
Non-pecuniary Incentives					
* Having fun programming is very important to me	66.2 %	75.0 %	67.7 %	76.9 %	61.5 %
* I contribute to Free Software because I believe information should be free[26]	57.8 %	66.7 %	63.4 %	61.5 %	46.2 %
* Personal exchange with other software developers is very important to me	42.2 %	38.9 %	46.2 %	34.6 %	46.2 %
* Lack of payment for my work in Linux projects is a significant inconvenience to me	1.9 %	0 %	2.2 %	0 %	7.7 %
Market Incentives					
* Improving my programming skills is very important to me	66.2 %	77.8 %	68.8 %	73.1 %	61.5 %
* Facilitating my daily work due to better software is very important to me	64.9 %	63.9 %	63.4 %	73.1 %	73.1 %
* Career advantages due to experience gained in Linux projects is very important to me	23.4 %	25.0 %	20.4 %	15.4 %	42.3 %
Both Market and Non-pecuniary Incentives					
* Improving the quality of the Linux Kernel in general is very important to me	53.2 %	69.4 %	51.6 %	69.2 %	53.8 %
* Gaining a reputation as an experienced programmer inside the Linux community is very important to me	22.1 %	27.8 %	23.7 %	11.5 %	34.6 %

DATA SOURCE: Raw data was downloaded from http://www.psychologie.uni-kiel.de/linux-study/writeup.html

[26] The origins of Open Source development date back to the 1970's and are rooted in the academic traditions of freely publishing research, subjecting work to peer review and sharing one another's discoveries. The early idealists, led by Richard M. Stallman, a revered programmer in this community, believed deeply that all software should be free of charge and that commercial software was immoral.

Table 2. Emergent Division of Labor

Emergent Roles of Linux Developers	Number of people	Total Number of Emails [NOTE 4] Sent to the Mailing List	% of Total Emails Sent
Project Leader	1	2,840 (the third highest number)	1.4%
Maintainers	121 [NOTE 1]	37,387 (including the project leader)	18.8%
Developers			12.3%
"The development team"	2,605 [NOTE 2]	20,563	10.3%
"The bug reporting team "	1,562 [NOTE 3]	4,216	2.1%

NOTE 1: Size estimation is based on the names listed in the MAINTAINERS file available in the source code file
NOTE 2: Size estimation is based on the names of email[NOTE 4] senders who wrote the word "PATCH" under the subject heading.
NOTE 3: Size estimation is based on the names of email[NOTE 4] senders who wrote the word "OOPS" under the subject heading.
NOTE 4: Source of emails: Linux-kernel email archive from June 1995 to August 2000.

Table 3. A Chronology of Stable Releases

Version	Starting-Ending Releases	First-Last Release Date	Release Frequency
1.0	linux-1.0.	12-Mar-94	
1.2	linux-1.2.0 : linux-1.2.13	6-Mar-95 : 1-Aug-95	14 releases in 5 months
2.0	linux-2.0.0 : linux-2.0.38	8-Jun-96 : 25-Aug-99	39 releases in 40 months
2.2	linux-2.2.0 : linux-2.2.16	25-Jan-99 : 7-Jun-00	17 releases in 18 months
2.4	linux-2.4.0-test1 : linux-2.4.0-test7	25-May-00 : 23-Aug-00	7 releases in 3 months

Table 4. A Chronology of Development Releases

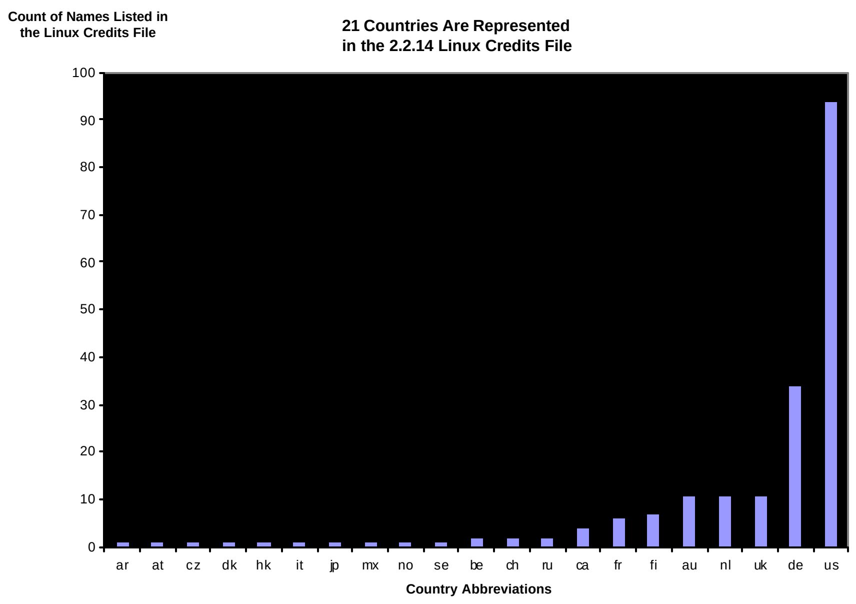
Version	Starting-Ending Releases	First-Last Release Date	Release Frequency
1.1	linux-1.1.13 : linux-1.1.95	22-May-94 : 01-Mar-95	83 releases in 11 months
1.3	linux-1.3.0 : linux-1.3.100	11-Jun-95 : 09-May-96	101 releases in 11 months
2.1	linux-2.1.0 : linux-2.1.132	30-Sep-96 : 22-Dec-98	133 releases in 27 months
2.3	linux-2.3.0 : linux-2.3.51	11-May-99 : 10-Mar-00	52 releases in 10 months
pre-2.0	linux-pre2.0.1 : linux-pre2.0.14.	11-May-96 : 05-Jun-96	14 releases in 1 month
pre-2.2	linux-2.2.0-pre1 : linux-2.2.0-pre9	28-Dec-98 : 20-Jan-99	9 releases in 1 month
pre-2.4	linux-2.3.99-pre1 : linux-2.3.99-pre9	14-Mar-00 : 23-May-00	9 releases in 2 months

NOTE: The releases are numbered using a hierarchical numbering system where the first number denotes a major version, and the second number gives the version tree in question. The stable releases have even version numbers (e.g., 2.0, 2.2, 2.4) and the development releases have odd version numbers (e.g., 2.1, 2.3).

Stable version and experimental version are two separate code trees for Linux. New features are tested in the development version first and then become included in the stable version. The stable version is where end users including business firms can find source code that is time-tested and proven. On average, the product cycle is on the order of weeks. The development version is where developers can experiment with advanced technology and try new ideas. At its peak frequency, there are as many as three new development kernel releases a day, a much shorter cycle than that of the stable version. In 1996 alone, the stable version had nearly 30 official releases while the experimental version had 80.

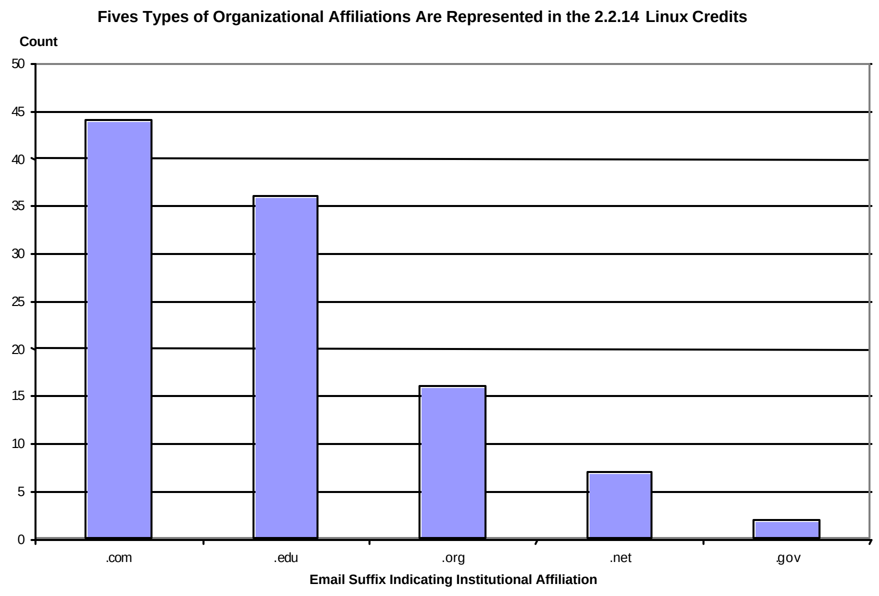
FIGURES

Figure 1. Nationality distribution of the people listed in the 2.2.14 Credits file



NOTE: ar = Argentina, at = Austria, cz = Czech Republic, dk = Denmark, hk = Hong Kong, it = Italy, jp = Japan, mx = Mexico, no = Norway, se = Sweden, be = Belgium, ch = Switzerland, ru = Russia, ca = Canada, fr = France, fi = Finland, au = Australia, nl = The Netherlands, uk = United Kingdom, de = Germany, us = United States.

<u>Figure 2. The distribution of organizational affiliation listed in the 2.2.14 Credits file.</u>


Fives Types of Organizational Affiliations Are Represented in the 2.2.14 Linux Credits
Count
.com
.edu
.org
.net
.gov
Email Suffix Indicating Institutional Affiliation