FROM A FIRM-BASED TO A COMMUNITY-BASED MODEL OF KNOWLEDGE CREATION: THE CASE OF THE LINUX KERNEL DEVELOPMENT

Gwendolyn K. Lee
Robert E. Cole

Haas School of Business
University of California, Berkeley

March 05, 2003

Abstract

We propose a new model of knowledge creation in purposeful, loosely-coordinated, distributed systems, as an alternative to a firm-based one. Specifically, using the case of Linux kernel development project, we build a model of community-based, evolutionary knowledge creation to study how thousands of talented volunteers, dispersed across organizational and geographical boundaries, collaborate via the Internet to produce a knowledge-intensive, innovative product of high quality. By comparing and contrasting the Linux model with the traditional/commercial model of software development and firm-based knowledge creation efforts, we show how the proposed model of knowledge creation expands beyond the boundary of the firm. Our model suggests that the product development process can be effectively organized as an evolutionary process of learning driven by criticism and error correction. We conclude by offering some theoretical implications of our community-based model of knowledge creation for the literature of organizational learning, community life, and the uses of knowledge in society.

I. INTRODUCTION

The objective of this paper is to develop a new model of knowledge creation in a distributed system, in contrast to a firm-based model. With the alternative model, we hope to address the possible uses of knowledge in society, particularly the use of knowledge that leads to further expansions of knowledge or innovation. How to utilize knowledge that is dispersed among many people is the question Hayek insightfully posed (1945). He observed that the critical problems in how to use widely dispersed knowledge are: (1) knowledge is not, and cannot, be concentrated in a single mind, and (2) no single mind can specify in advance what kind of practical knowledge is going to be relevant, when and where (cf. Hayek, 1945, 1982, 1989; Tsoukas, 1994). While Hayek proposed the market, where knowledge is embedded in prices, as a solution to these problems, much research on knowledge-based organizations, however, focused on hierarchy (Kogut & Zander, 1992; Nonaka & Takeuchi, 1995; Teece, 1998). Unfortunately, the focus on hierarchy missed a key opportunity to address those problems associated with the use of distributed knowledge.

By contrast, we address the use of distributed knowledge for the purpose of innovation at the community level, beyond the boundary of a firm. In comparison, we do not, however, fall back on the price system ′a la Hayek, because the market is a static solution based on a strong assumption of equilibrium. Therefore, it does not explain the processes leading to knowledge expansion. The conventional firm-based model of knowledge creation also has several limitations in addressing knowledge creation processes, given the advancement of information technology and the intensification of globalization. For instance, the advent of the Internet and web-based technologies has enabled specialized communities to convene, interact, and share resources extensively via electronic interfaces. The highly distributed nature of knowledge as such presents a set of challenges that force organizational analysts to reconsider the knowledge creation processes inside the firm.

In this paper, we develop a community-based model of knowledge creation where the community's organizing principles seriously challenge the traditional assumptions of the firm-based model. In developing our community-based model of knowledge creation, we adopt an evolutionary framework (Campbell, 1960) and analyze the role of criticism and critical evaluation as espoused by Karl Popper (1973) as a key driver in the evolutionary process. Our analysis is based on the case of Linux kernel development and our goal is to explain how a group of distributed individuals, dispersed across space, time, and organizational boundaries, organize themselves to create a useful product of high quality. In the next section, we review foundational literature on knowledge creation and outline our community-based model of knowledge creation. In section III, we describe the research setting, methods, and data to develop the case of Linux kernel development. In section IV, we analyze the case of Linux kernel development in detail. In addition, we will compare our community-based model against the firm-based model, and then against a model of traditional/commercial software development. In section V, we bring together the theoretical contributions of our community-based model of knowledge creation. Finally, we offer some suggestions for future research and discuss the generalizability of the model in section VI.

II. MODELS OF KNOWLEDGE CREATION

The foundational literature in knowledge creation ill prepares us for a world of knowledge creation through distributed cognition and geography (Nonaka & Takeuchi, 1995; Argyris, 1993; Prahalad & Hamel, 1990; Levitt & March, 1988; Winter, 1987; Teece, 1986;

Nelson & Winter, 1982; Penrose, 1959). This is not surprising since much of the literature was formed before the Internet became a true force in knowledge collaboration via electronic systems designed to help work teams increase quality and utilization of intellectual assets. 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 take place at the level of a firm or a set of firms. A firm is seen as the repository of knowledge and the physical locus of its creation and deployment because it provides the social, political, and economic context under which knowledge is constructed. Even if the researcher focuses on the individual such as Argyris (1993) does, it is in the context of producing learning for a firm. When a firm or a network of firms is the unit of analysis, we naturally anticipate that a firm will treat the knowledge created by its employees as a 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; Brown & Duguid, 2000).

Second, the foundational scholarly literature on knowledge creation assumes face-to-face interactions 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 knowledge creation (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, where organizationally as well as geographically dispersed near strangers can collaborate.

Third, the foundational literature, by relying on a firm as the unit 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-managing) 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 well-known 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. However, as outlined in the very beginning of our discussion, the two critical problems derived from the use of distributed knowledge presume that a single mind is unable to survey, or even to control a firm's knowledge.

Quite the opposite, in the community-based model of knowledge creation that we develop, the community's organizing principles marginalize the critical assumptions of the firm-based model. Our proposed model includes both the norms (the nature of rules) and the forms (the structure of exchange) of knowledge creation, in a community of practice embodied in product development. The community-based model pays special attention to how problem solving drives the evolutionary process of knowledge creation and generates structure in a community setting. We emphasize the role of criticism and error correction in the process of knowledge creation where innovations are continuously generated, selected and retained to produce innovative and high-quality products.

In Table 1, we summarize the major differences between the firm-based versus the community-based model of knowledge creation. The three assumptions of the firm-based model discussed above are violated when (1) the assignment of intellectual property rights promotes trust building and knowledge sharing; (2) when the membership is open and consequently the size of a community is much larger than that of a firm; (3) worker incentives and motivations shift from those of employees to those of volunteers, and unlike in a firm setting, there isn't an authority relationship to regulate the behavior of community members; (4) individuals are organizationally as well as geographically dispersed; and (5) the knowledge-creation platform is based on a many-to-many communications technology. These five conditions together raise a set of serious challenges to organizational analysts above and beyond the two critical problems posed at the very beginning of the paper.

<u>Table 1. The Community-based Model vs. the Firm-based Model of Knowledge Creation</u> HERE

With the development of the community-based model, we would like to explain how a community of distributed volunteers, under conditions where firm-based mechanisms of hierarchical control and authoritative command appear problematic, achieves quality (product reliability) while maintaining a highly innovative process. The concept of "communities of practice" as set out in Lave and Wenger (1991) points us in the right direction. In principle, there is no reason why communities of practice should be limited to those within the same firm. However, most examples of communities of practice used in the organizational literature have tended to be intra-organizational and focused on emergent groups in existing organizations, including the focal firm and its suppliers and customers. A case in point is Brown and Duguid (1991) where communities of practice are studied as the unit of analysis for understanding knowledge inside the firm. So, our community-based model of knowledge creation contributes to this literature by showing how processes of knowledge creation are enacted in communities of practice beyond the boundary of the firm.

Recent development in another related literature has extended the scope of the traditional assumptions of firm-based models in ways that favor the creation of knowledge products through distributed cognition and geography. Tsoukas (1996) proposed to view a firm as a distributed knowledge system where a firm's knowledge is the indeterminate outcome of individuals attempting to manage the inevitable tensions between normative expectations, dispositions, and local contexts. He argued that a firm's knowledge is distributed not only in a computational sense (Hutchins, 1993), or in Hayek's (1945) sense that the factual knowledge of the particular circumstances of time and place cannot be surveyed as a whole, but also in a radical sense that nobody knows in advance what that knowledge is or need be (Tsoukas, 1996: 22).

Although we certainly agree with Tsoukas that a firm can be viewed as a distributed knowledge system, we find this view unsatisfying because Tsoukas does not go on to specify in detail how knowledge is created in a distributed system other than that a firm's knowledge is emergent and never complete at any point. All processes leading to expansions of knowledge, as Campbell (1960) argued, have three necessary conditions: a mechanism for introducing variation, a consistent selection process, and a mechanism for preserving and reproducing the selected variations. Essentially, knowledge creation processes involve iterations of a "blind-variation-and-selection-retention" cycle that is evolutionary in nature (Campbell, 1960: 380). However, as Kogut (2000: 408) pointed out, "organization by firm is variety reducing." As a repository of knowledge, firms are social communities that permit the specialization in the creation and replication of partly tacit, partly explicit organizing principles of work (Kogut, 2000). That is, a firm's organizing principles make it an efficient vehicle for the accumulation of specialized learning, but not a generator of variety in innovations. So, this research points us in the direction of seeking a better understanding of knowledge creation beyond the boundary of a firm.

Furthermore, the prominent feature of our community-based model of knowledge creation is a set of rules and structures that encourages critical evaluation of existing knowledge, innovation, and rapid elimination of error. In his comparison between markets and firms, Kogut (2000) suggested that an economic network, which is a collection of firms, each ensconced in an identity that supports specialization and a dynamic of learning and exploration, offers the benefit of both specialization and variety generation. More importantly, the structure of a network is an expression of competing and evolving rules that guide the behaviors of interacting entities. This insight leads us to focus on the emergence of structure reflecting the operation of organizing principles that act as "genetic rules" as we develop a new model of knowledge creation based on a community rather than a firm setting.

Our model development contributes to how researchers can better understand the "motors" - the event sequences and generative mechanisms – of development and change in a community setting (Van de Ven & Poole, 1995). In this setting, teleological, dialectical, and evolutionary motors interact, leading to the development of a complicated, tri-motor theory of knowledge creation. With the norms and the forms of knowledge creation, we explain in this paper how dialectics (i.e., criticism and critical evaluation) drive an evolutionary process in a purposeful, loosely-coordinated, distributed system, which has an objective to produce high quality, knowledge-intensive goods.

III. RESEARCH SETTING, METHODS AND DATA

Research Setting

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, the Linux operating system development project created one of the Open Source communities.[1] This is not to say that the project leader preplanned for the high level of participation that ensued, but he intentionally reached out to other participants on the newsgroup for their help. The community 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. 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 these others is much smaller. 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."

Shortly after the emergence of electronic commerce via the Internet in the mid-1990s, the demand for web serving and networking technologies by new and incumbent firms increased rapidly. 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.

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).[2] Linux runs on computer systems of small networks, which Internet service providers and university computer labs use, as well as on 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. [3] In this sense alone, Linux must be regarded as a success and this high utilization rate relative to other products suggests high quality, particularly as measured by reliability (see Cole & Lee, 2002).

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, 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

[1] The Open Source definition is available at http://www.opensource.org/osd.html.
[2] The Internet Operating System Counter, http://www.leb.net/hzo/ioscount/index.html.

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.

As is well known by now, the original creator of Linux is Linus Torvalds, who was a 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. 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. Torvalds (1992) released the "Free minix-like kernel sources for 386-AT" free-of-charge in October 1991 to the newsgroup and solicited other programmers to contribute computer code that they would add to the Linux source code.

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. To date, thousands of self-motivated contributors across different countries, who wanted to try their hands on cutting-edge computer science, have participated in the development of Linux. Seemingly, a 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.

[3] http://www.linux.org/info.

Methods

Our research goal is to improve the theoretical understanding of the knowledge creation process. To insure that we build our analysis on everyday practices, we applied inductive, rather than deductive, reasoning to allow for modification of concepts and relationships among concepts. Following Znaniecki's (1934: 261-262) steps of the analytic induction process, we constructed a model based upon an individual case representing the Open Source phenomenon, in contrast with the firm-based model of knowledge creation.[4] We also followed Becker's (1958) suggestion to include a natural history of the various stages of the analytic induction process in arriving at conclusions, so that the readers are able to judge the adequacy of the procedure and the evidence for conclusions. We have created original terms and novel classifications in this process and they mainly come from our observations and analyses. More specifically, we categorized the developers into four types according to their roles and contributions made to the community. Additionally, we identified a two-tier structure according to how tasks naturally emerged in the community.

Data

Our study is grounded in archival data analyses, on-line research publications, and observations of how the technology has evolved. Appendix A shows a list of the data sources that we used. Among these data sources, 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

[4] Znaniecki lists several steps in the analytic induction process: 1) develop a hypothetical statement drawn from an individual instance; 2) compare that hypothesis with alternative possibilities taken from other instances.
[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

data because the Linux kernel development project had stabilized by version 2.2; version 2.2 was developed between 1999 and 2000. Since then, the more exciting developments for the Linux operating system take place outside the kernel (Torvalds, 1999: 111).

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. 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.[6]

An equally important artifact is the development activities archived in the Linux-kernel mailing list, which was created for the purpose of discussing kernel development.[7] 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. Using the weekly Linux-kernel email archive for 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]

In addition, we examine the developers' demographic distributions, 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

[6] Only 28 maintainers are listed in the Credits file
[7] http://www.uwsg.indiana.edu/hypermail/linux/kernel/

Germany to the Linux-kernel mailing list (see Appendix B). The period of data collection was between February 2000 and April 2000.

We arrived at four mechanisms critical to knowledge creation by generating categories from our initial data collection and analysis and then refining them through a series of iterations. First, we observed, from the Linux-kernel mailing list discussions, that each release of a new version of the source code starts a cycle of software development. In each cycle, we estimated the size of the development community and analyzed the developers' roles and intensity of their online activities. In addition, we analyzed the structure of the source code as well as the distribution of developers and lines of code across each module. Then we studied the structural evolution of the Linux development community by examining the changes in size and code structure across time. We found that the size of the community increased by four fold between 1995 and 2000, and the structure of the Linux source code became modularized. Over the same period, we also found that through our analysis of developers' email suffixes, the demographic distribution of the community became much more diverse, as measured by the dimensions of nationality and institutional affiliations.

A natural question that directed our subsequent analyses was: Why do so many computer programmers across different countries and organizations participate in the Linux kernel development? We then analyzed how respondents of the Linux-kernel survey ranked several statements revealing their motivations and reasons why they participate in the development project. Another question that followed was: How do these many motivated developers coordinate their effort such that the system is not chaotic? With that in mind, we studied the norms in the community and identified areas where rules apply and the consequences of non-compliance. Finally, we analyzed the speed of development by tracking how quickly emails are

[8] As of August 26, 2000, there were a total of 199,374 emails archived in the mailing list.

responded and how frequently newer versions of the source code are released.

All along throughout the data generation phase, we grouped and regrouped the findings to identify the mechanisms that underlie the Linux development process. After several iterations, we arrived at four mechanisms by which (1) the knowledge building blocks become widely shared (intellectual property licensing), (2) a diverse set of computer programmers across organizational and geographical boundaries becomes volunteers in the Linux community (incentives to contribute) (3) the effort of numerous developers becomes useful products without hierarchical control (coordination under conditions of uncertainty), (4) unplanned contributions become a high quality product (quality in processes of innovation).

IV. CASE STUDY: THE LINUX KERNEL DEVELOPMENT PROJECT

Existing scholarly accounts of the Open Source phenomenon (Lerner & Tirole, 2000; Markus, et al., 2000; Moon & Sproull, 2002) have adopted the perspectives of labor economics, virtual organization, and distributed work, respectively, to explain two questions that remain puzzling: (1) what motivates software developers at many different countries and organizations to volunteer their time and effort in Open Source projects and (2) how are on-line activities of distributed groups managed in the absence of employment or fee-for-service contracts. However, given the explicit innovative objectives of product development projects, it is remarkable that existing scholarly accounts of the Open Source phenomenon fail to address the processes by which seemingly ad hoc contributions turn into products of high quality.

In addressing the knowledge creation processes in communities of practice, we present our case study in three parts. First, we will discuss the legal and cultural norms of knowledge creation in the Linux development community that enable not only open sharing but also criticism and critical evaluation through which high quality is achieved while maintaining a highly innovative process. Next, we will examine the form of knowledge creation and focus on how the norm of critique leads to the emergence of structure. Finally, we compare and contrast the Linux development case, first with the firm-based model of knowledge creation, and subsequently with the traditional/commercial model of software development.

1. The Norms of Knowledge Creation in Communities of Practice

The management of intellectual capital and in particular, the treatment of intellectual capital, holds the key to securing a continuing supply of knowledge assets (Boisot, 1998: 255). To be sure, when key knowledge contributors are volunteers, not employees, the issue of how intellectual property gets treated and leveraged involves setting up different rules of the game under which volunteers are motivated to contribute.[9] While existing accounts of the Open Source phenomenon study the mechanisms sustaining the norm of open sharing of intellectual properties, there has been no exploration of how these mechanisms maintain the quality of the shared intellectual properties. In our subsequent analysis, we focus on intellectual property licensing and culture, not only as important control mechanisms in regulating the norm of open sharing, but also as key quality assurance mechanisms in eliminating errors in shared intellectual properties through the norm of critique.

Intellectual property licensing, such as an Open Source software license, can set up a legal environment by laying out the rules of the game such that motivated contributors can trust their intellectual properties with individuals with whom they have not had prior personal contact. For example, the terms of an Open Source software license stipulates the requirement that any change or improvement to the source code is to be made available to the public. Additionally, modifications of existing source code must be distributed under the same license as the original software. Through the interplay between the existing system of copyright and the Open Source software license, the norm of open sharing can be enforced legally. As a guard against misuse of the shared intellectual properties, these licensing terms are credible because existing laws of intellectual property rights permit the code contributor, who in many cases is the copyright holder, to pursue legal actions in the court of law against deviant behaviors away from the community norm.[10]

At the same time, the norm of open sharing is maintained socially. As Markus and associates (2000) observed, the Linux development community, like other Open Source projects, use culture as a social control mechanism to manage membership, make decisions, monitor individual performance and behavior, and sanction developers who deviate from group norms. For instance, one of the community protocols is a strong cultural norm to properly cite authors whose work is being extended or borrowed. As an important part of the culture, this protocol shows how much this community respects the contributions and intellectual properties of its individual 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 a description of their respective contributions (see Appendix C for an example).

[9] However, there is little organizational behavior research on the comparison of volunteers and employees, except Pearce (1983) and Laczo & Hanisch (1999) that addressed issues of sufficiency-of-justification effects (cf. Staw, 1976; Deci, 1975; Festinger, 1961) and withdrawal behavior, respectively.
[10] However, "copyleft" has yet to be challenged in the court of law (McGowan, 2001: 252)

Indeed, legal and cultural norms of open sharing are critical to the management of intellectual property in a community setting. However, making sure that the quality of the shared intellectual properties is high is an equally important objective. In addition to reserving a legal threat against misuse of intellectual properties, the Open Source software license by its very openness plays an important role in eliminating the concealment of error. The terms of the license make knowledge public and that is a major requirement for criticism to drive the learning process (Popper, 1989: 260). Similarly, culture is not merely a social control mechanism. More significantly, culture, as manifested in the activation and channeling of criticism and error correction, serves as an important function in the process of innovation and learning in a distributed system.

We observe criticism operating as a cultural norm, which increases the likelihood of uncovering error. Consider Torvald's vision of the peer review process:

> … 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… <u>The point of open development is that people see what's going on</u>… <u>You want to have random people just see small updates - because they will often catch silly mistakes.</u> (quoted from an email sent by Linus Torvalds to the Linux-kernel mailing list in August 1999)

Two points are highlighted in this statement. First there are specific modes of peer review that will increase criticism and through an iterative process create better knowledge and quality. Second, open development is all about being open! That is, it is about being public and exposing new input to all interested eyes. In Popper's language, Torvalds is encouraging "open critical discussion" that moves us closer to the "truth" (Popper, 1989: 262).

The norm of critique underpins the development process leading to knowledge expansion. In the Linux development community, we observe a peer review process as a structured approach to generating criticism of existing versions, evaluating those criticisms and eliminating "error" while retaining those solutions that cannot be falsified. In the process of peer review, one builds one's reputation partially by creating useful solutions and partially by sound critical evaluations of the work of others. On one hand, the quality of prior submissions becomes a currency, which developers exchange for the community's attention on their next submission. On the other, criticism serves as a signal to individual developers and the development community.

Specifically, the peer review process is the key approach in connecting the process of generating variations to that of selecting code. Typically, the project leader and the maintainers select code for inclusion based on technical merits. When technical merits are not immediately evident, 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 documented in the Linux FAQ. The stated criteria of patch acceptance are as follows: the code has to (1) appear "obviously correct" to the project leader, (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.[11]

More importantly, the peer review process meets a key condition for the growth of knowledge as outlined by Popper (1985: 172). Through technologically-mediated communications, developers can conveniently and quickly exchange, compare, search, and discuss any change because these changes are publicly archived in the mailing list. Among the email discussions in "the development team," we found a 7 to 3 ratio between responses (review) to patch submissions (generation). That is, for every patch submitted, there are 2.35 responses

[11]The "useful solution" being created by developers are primarily "patches." The stated criteria of patch acceptance are listed in the the Linux Frequently Asked Questions (FAQ), under "How do I get my patch into the kernel?"

on average. Here we see criticism and error correction serving as a driving force for Linux development.

2. The Forms of Knowledge Creation in Communities of Practice

Applying Kogut (2002), we can say that criticism and error correction operate as genetic rules that generate the structures of the development community – the parallel code structure and the two-tier task structure. The parallel code structure plays a significant role in maintaining the three necessary conditions of knowledge creation suggested by Campbell (1960). The stable and the experimental versions are two separate code structures (trees) of Linux running in parallel. The stable tree focuses mainly on fixing bugs, while the experimental one serves as a test bed for new features. Once an experimental version incorporates sufficient new features and becomes sufficiently reliable through bug fixes and patches, it is then renamed and released as a stable version. The experimental tree is where innovative variations and experimentations are generated, tested, and selected to become a part of the stable tree where further refinement and improvement are conducted. On average, since the first release of Linux, there has been one new version of the source code released to the public every week. Table 2 shows a chronology of the official code release frequency for the stable version and the experimental version of the Linux kernel source code.

Table 2. A Chronology of The Parallel Code Tree Release Frequency HERE

The parallel code structure is a clever structural device for channeling these development and improvement processes in ways that allow, indeed encourage, experimentation to occur and errors to be freely made without endangering the larger community's ongoing use of the stable version. This is an ideal device for encouraging knowledge creation as an improvement process driven by criticism. At any given time, participants view the existing versions, in effect, as hypotheses of what works best at that time but they expect that their subsequent experience, and those of others using Linux, will reveal flaws and weaknesses that need correcting.

The experimental version, in particular, can be seen as a vehicle for generating and discussing error as visible public property and making the most of error as a learning device. As such, it personifies Karl Popper's view of how mistakes should be treated (Popper, 1989: 260). As shown in Table 2, the experimental tree has a much higher release frequency than the stable tree. 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, which is in the order of weeks. In 1996 alone, the stable version had nearly 30 official releases while the experimental version had 80. The startling frequency of the iterations of the stable and experimental versions must be regarded as a central feature of Linux's process of knowledge creation and improvement. Underlying these iterations is a fundamental process of criticism in the Popperian sense in which at any given time existing versions serve as a testing ground for criticism and improvement (Popper, 1989: 260-261). We see demonstrated in these activities a susceptibility to revision and repeated efforts at falsification, one of the hallmarks of the scientific method (Popper, 1985: 137).

The second structure that emerged is a two-tier task structure. By observing the Linux developers' tasks, roles, and levels of involvement between 1995 and 2000, we find that these developers naturally sort themselves into a two-tier structure thereby enabling the Linux development project to cope with the complexity of task coordination under conditions of uncertainty (see Table 3). This two-tier structure consists of a small core and a large periphery.

The core consists of a project leader and hundreds of maintainers, while the periphery has thousands of developers organized into two teams, namely "the development team" and "the bug reporting team."

Table 3. A Two-Tier Structure HERE

In the observed two-tier structure, the core relies on the periphery to generate patches of computer code and bug reports. The generated variations of the source code in the form of patches and bug fixes are tested and reviewed by others in the periphery. Peer developers then provide evaluations as well as suggestions to others' work-in-progress so as to encourage improved variations. Subsequently, the core selects and retains among the variations to produce an official release, which then goes through more cycles of tests, variation generations, and improvements. As such, the tasks of variation generation and review are separated from those of selection and retention.

The roles emerge in the process of performing tasks, as opposed to being planned or someone being assigned to carry out a task. For example, in the core, the project leader as well as the maintainers, who are responsible for various subsystems, perform the selection of source code and decide which code should become included in the next kernel for official release.[12] In the periphery, we found, over the five-year period, 2,605 people on "the development team," which adds features and fixes bugs. They electronically submitted to the Linux-kernel mailing list, patches of source code and emails to discuss the code. Over the same period, we also found 1,562 people on "the bug reporting team," which reports, documents, or characterizes a bug.

[12] The 2.2.14 MAINTAINERS file lists 121 maintainers in charge of 132 subsystems. Some subsystems have co-maintainers and some maintainer watches over more than one subsystem.

They submitted to the Linux-kernel mailing list the failure messages, which are automatically generated when the kernel detects a problem during use. In the bug report, they also provided some description of how to recreate the problem. In addition, we found that some developers on "the development team" overlap with those on "the bug reporting team." Forty-nine percent of "the bug reporting team" also performed the tasks of "the development team," while 29 percent of the development team performed tasks of the bug reporting team. So, a developer generally plays different roles and performs multiple tasks in the periphery.

In our analysis of emails by roles, we found on average that peers in "the development team" contributed more frequently to review (68% of all responses) and generation (81% of all the patches submitted) than the core developers (32% and 19%, respectively). This shows that the peers in the periphery are more active than the core developers in the tasks of generation and review. However, we also observed that the core's interaction with the periphery increases as the degree of exchange increases.[13] At the first level of response to a patch submission, the core developers contributed to 31% of the discussion, but at the fourth level of response (the longest chain of response observed), the core developers contributed 68% of the discussion. This is evidence confirming the selection and retention function served by the core developers and it shows that the decision-making process is highly social and stratified.

We see here the norm stressing the importance of critical evaluation manifested in the two-tier task structure. The norm is implemented through the aforementioned peer review process, which provides a systematic method for identifying and criticizing error or weakness in existing versions and contains a method for correcting these errors or weaknesses. Peer developers in the periphery take the variations that other developers generate with the source code and perform various "stress tests" on them. 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. Then the developers in the periphery provide feedback and criticism as key input to the developers in the core to select specific variations leading to the next official release. Indeed, criticism operates as a genetic rule in the generation of the two-tier task structure that is fundamental to error detection and correction.

More specifically, developers use the word "PATCH" in the email subject heading to signal that they are proposing a solution with a patch of code, or discussing/criticizing a patch. The proposed patches often add new features to the operating system and fix problems or bugs. Developers also use the word "OOPS" in the email subject heading to report a bug, or to fix a bug. [14] The content of the “OOPS” email often includes bug identification, characterization, and ideas for elimination.

3. <u>The Community-based Model As An Alternative to the Firm-based Model</u>

How to increase knowledge sharing among employees is indeed a key challenge as researchers have shown in their case studies of groupware use in professional services firms; they find that knowledge sharing is an important strategic goal, yet fairly difficult to achieve (Orlikowski, 1993; Davenport, 1996, 1997). Unfortunately, firm-based knowledge creation efforts are typically constrained by the number of employees that a firm can hire, the quality of the extant employee pool, knowledge hoarding, and incentives that reduce motivation to share across units. In our analysis above, we have shown how our community-based model of knowledge differs significantly from the firm-based model.

[13] We measure the degree of exchange by counting the length in the chain of responses.
[14] "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

First, the norms of knowledge creation are different in how intellectual property rights can be assigned in novel ways that promote trust, insure the sharing of knowledge, and reveal error. Also, in contrast with the community-based model of knowledge creation, the firm-based model does not rely strongly on an evolutionary process driven by criticism and critical evaluation, which we described earlier as the cultural norm for Linux. MacDuffie (1997) shows strong variation in the willingness of firms to identify error and in their mode of conceptualization and in how they delineate the scope of consequent problems and sources of possible resolution. He further shows how problems of developing common language and common standards across units thwart criticism and problem solving activities. While some firms, at great cost, can develop capabilities that minimize these problems, the Linux project seems ideally designed to maximize criticism from the start.

In the case of commercial software, the knowledge creation efforts in the firm-based model tend to grind to a halt after the initial product launch because attention subsequently shifts to services and to other projects. Most commercial software companies release their products and/or follow-up upgrades only every few years. Furthermore, the disconnect between users and developers leads to great difficulties in being able to duplicate much less correct error. Among the most frequently reported reasons for failure among commercial software firms are the codes no-fault found (NFF) and cannot duplicate (CNN).[15] Furthermore, to be associated with error is often an impediment to reputation and career success. Thus, there are great incentives in commercial software firms for individual employees to conceal rather than reveal error or weakness in their work. Overall, in comparison with commercial software, Linux is a

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.

continuously evolving product with much higher update frequency, a frequency that rests on systematic and continuous criticism of existing versions and a rapid feedback process to eliminate error.

Second, in the community-based model of knowledge creation, the norm of critique generates the structure of the development community. The two-tier structure, in particular, accommodates scale better than a hierarchical structure typically found in a firm. However, this finding seems to stand in sharp contrast against the predominance of hierarchic systems common in a firm setting where complexity is high. The correlation between scale and response time is often understood to be positive in a hierarchy. The larger the scale is, the longer the response time. For example, in his influential 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.

Nevertheless, we observe that for Linux the positive correlation between scale and response time holds only in the core, but not in the periphery. More specifically, adding more developers to the core may initially increase the speed of selection but later lead to diminishing returns as complexity or communications cost increases, or even to negative returns if conflict cannot be resolved. In contrast, adding more developers to the periphery may increase the probability of generating more variations, but may not prolong response time. We suggest that the correlation between increase in team size and delay in response time is contingent on the task of the developer. In the periphery, which is the larger of the two, the two-tier structure

[15] We are indebted to Greg Watson, software quality consultant and former President of the American Society for Quality, for pointing this out.

accommodates increasing scale better because the correlation between scale and response time is not necessarily positive. This argument is consistent with Herbert Simon's "near decomposability," a key property of hierarchies that helps organizational analysts understand "how the information needed for the development or reproduction of the system can be stored in reasonable compass" (Simon, 1996: 216). In a hierarchical structure, the main function of middle layers is to review subordinates' work and pass on to the next layer. The review tasks are similar to the ones performed in the core. So, we reconcile this potential conflict by viewing the two-tier structure as a flat hierarchy in its simplest form with the top and middle layers collapsed into one.

Finally, our emergent model extends the firm-based model by emphasizing the use of a knowledge-creation platform that allows many-to-many digital communications. The Internet, for example, provides a knowledge-creation platform with which users conveniently and cost-effectively transmit digitized information such as software code, text, picture/image, voice, and video. Subsequently, these digital transmissions become objects of discussion and artifacts of knowledge creation. On the other hand, television, radio, and other broadcast media are one-to-many, while face-to-face interactions and dialogues over the telephone/two-way radio are one-to-one communications. A few firm-based case studies have illustrated the use of two-way radios by a community of field technicians to share and weave stories about their experiments with a machine (Orr, 1990a, 1990b, 1987a, 1987b). In comparison, the Internet provides unrivaled reach and richness for a large-scale community to share knowledge (see Castells, 2001; Dreyfus, 2001; Selvin, 2000). It is part of the historical evolution of human communications and knowledge sharing. This evolution is based on enhancing face-to-face interactions initially with one-to-many communications, which historically began with drums, signal fire, and later the invention of the printing press, and has now arrived at many-to-many communications. Because the Internet technologies became popular after the firm-based model was developed, we focused on the benefits of such technologies in our model.

In the case of commercial software, the model of knowledge creation adopted remains largely firm-based, although software development firms do emulate three of the five conditions that we discussed through increasing the use of the Internet (the condition of a many-to-many, digital, knowledge-creation platform) the involvement of application developers from multiple firms (the condition of distributed cognition and geography) and leveraging leading customers (the condition of volunteer incentives) to assess product features and identify bugs.

In contrast to these similarities with the traditional/commercial model of software development, our community-based model provides a novel approach to the licensing of intellectual property rights (the condition of Open Source licensing) and the size of the development community (the condition of open membership and increasing scale). The source code of Open Source software is distributed without charging a royalty or other fee, while the source code of commercial software is mostly proprietary.[16] Consequently, holding every thing else equal, the scale of an Open Source development project can be much greater than that of a commercial one.[17] This novel approach to intellectual property licensing and scale represents a different development process. This approach is consistent with Raymond's argument that the Linux development model is unique because the product was created in a "bazaar," by a large development team that exchanges work-in-progress frequently, not in a "cathedral" by small, isolated development teams that do not exchange any work until its completion (1999,

[16] A list of distribution licenses approved by the Open Source Initiative is available at http://www.opensource.org/licenses/index.html.
[17] "The bigger product units such as Office, Windows NT, and Windows each have between three and four hundred people." (Cusumano & Selby, 1995: 50)

Raymond's terms are in quotes). Although commercial software development firms use "daily build" to simulate the "bazaar" in exchanging work-in-progress, this development process typically involves only the programmers, not the users. As shown in the Linux case, involving users heavily in product and knowledge development process significantly increases the sources of idea generation and variations for selection and retention.

V. THE COMMUNITY-BASED MODEL OF KNOWLEDGE CREATION: AN EVOLUTIONARY PROCESS OF LEARNING

Our aim has been to use the Linux case to show how the process of knowledge creation takes place beyond firm boundaries and how it needs to be understood as an evolutionary process of learning driven by criticism. The evolutionary process can be seen as one of generation followed by error identification, detection, and rejection. Criticism as a mechanism for learning is supported by Linux's norms and structural arrangements, through which innovations are continuously generated, selected and retained to produce products of high quality.

Schon summarized this evolutionary process in his description of a learning organization:

> "Like the individual craftsman, the collective has a theory-in-use implicit in the norms, strategies, and assumptions that govern its regular patterns of task performance. [The] theory-in-use may be inferred from the evidence of intelligent action, especially from *the detection and correction of errors*" (Schon, 1983b: 118, emphasis in italics added).

The rapid evolution of source code in a parallel structure leads us to characterize the Linux kernel development project as a self-designing system. The image of organizations as self-designing systems blends the image of repository with that of culture (see Weick & Westley, 1996; Weick & Berlinger, 1989; Eccles & Crane, 1988; Weick, 1977; Hedberg, et al., 1976 on self-designing systems). On one hand, as repository, the system accumulates built-up knowledge as trust-worthy public property, meeting a key condition for learning from error posited by Karl Popper (1989: 260). This knowledge takes the form of cultural artifacts, which include "principles and maxims of practice, images of mission and identity, facts about the task environment, techniques of operation, stories of past experience which serve as exemplars for culture action" (Schon, 1983a: 242). On the other, as culture, the system captures clearly the learning that has taken place and provides a fertile ground where participants, through a process of criticism and problem solving, can better retest, modify, and/or affirm the learning. Combining these two images, the Linux self-designing system generates cultural artifacts such as source code and critical evaluations that function as the locus of learning.

The existing literature on learning as cultural processes is slim (Weick & Westley, 1996: 442). Culture, as Weick and Westley suggested, is embodied in artifacts, the material objects a group produces. Secondly, culture is embodied in the language, the words, phrases, vocabularies, and expressions which individual groups develop. Lastly, but most importantly, culture is embodied in coordinated action routines and predictable social exchanges. Open Source licensing and criticism as a cultural norm together provide exactly the legal and social environment for such routines and social exchanges to take place. Our model of community-based knowledge creation contributes to this literature a better understanding of learning based on criticism of the status quo where such criticism is inherent in the culture.

As observed earlier, criticism of the status quo is inherent in the Linux culture. First, the learning process uses the cultural artifacts as an educational tool. On one hand, the publicly archived criticisms help given individuals to learn from their peers' critique on how to improve their next submission. On the other, they serve as documented texts to train still other developers observing the peer review process. As developers learn from their own and others' prior successes and failures, they can sort themselves into tasks appropriate to their skills, move up to more challenging tasks, and/or generate better variations of the source code.

We observe that among the submitted patches and bug fixes generated by the periphery, the core selects conservatively what to retain in 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. We see here a stream of continuous challenges to the status quo being subject to critical evaluation with most being eliminated. In that sense, the efficiency of product development seems to be rather low because 77 percent of the contributions appear to be wasted.

However, from a cultural perspective, the rejected submissions provide a fertile ground for developers, both the contributing author and other community members, to learn what the required acceptance criteria are for further improvement. To the developers whose contributions are rejected, their submissions are cultural artifacts that serve as educational tools for the next round of peer review. As Popper (1985: 172) among others has noted, it is possible and indeed extremely important to learn from our mistakes. To other developers who follow the review process, the discussions that take place in the peer review are also cultural artifacts that record examples of other developers' trial and error.

Second, criticism of the existing status quo, which is the closed source model of software development based on privately-held intellectual properties, is profound in the Linux culture. Consistent with Karl Popper (Popper, 1972: 25, 260), we agree that criticism of the status quo commercial software constituted a major motivational force for the developers of Linux. Linux developers were engaged in an enterprise of "imaginative criticism" (Popper, op cit., p.148) in which they transcended their existing environment to create something new beyond their experience, a new world in which shared intellectual property provides a viable option to privately-held intellectual property. This criticism is at the heart of the Open Source movement, as geographically dispersed individual programmers, acting with common purpose, solidarity and continuity, promote these changes.

However, contrary to Popper, but consistent with Feyerabend's critique of Popper (Feyerabend, 1978), this criticism does not simply lead to hard-won scientific knowledge through a process of error elimination. Rather, it also serves as an ideological mobilizing force among hackers and programmers who become the developers of Linux. They internalize the criticism of the status quo, or in Popper's term (Popper, op. cit., p.286), become "consumers" of the criticism, with their constant efforts to further improve their new world alternative. As an example, the "Linux-kernel survey" shows a consistent agreement across all respondents that one of the main reasons driving their participation is because they believe "information should be free." It is in this context that many of these programmers saw creating an Open Source alternative to existing commercial software as a kind of crusade. Although we cannot demonstrate the matter empirically, it is a reasonable hypothesis that criticism of Microsoft's monopoly over operating systems was a major motivational driver.

In summary, Open Source licensing makes knowledge public; the criteria for achieving high reputational status in the Linux community provide incentives for innovation and criticism; the parallel code structure provides structural opportunities and incentives for continuous experimentation; the two-tier structure combined with a peer review system enables a systematic evaluation, error correction, selection, and retention; and, perhaps most importantly, cultural norms reinforce the structure of the community. Together, these characteristics provide the basis for the evolutionary development of Linux. They also meet the conditions for the growth of knowledge suggested by Karl Popper. This confluence is not surprising since Popper was interested in accounting for the development of scientific knowledge and Linux has been structured, perhaps unconsciously, to mimic many of these same conditions.

The broader organizational implications of the community-based model of knowledge creation are important to the literature of organizational learning. First, as discussed at the end of Section IV, the consequence of having users/consumers more involved in direct production and evolution of the product they use over time is the increased variety of input into the selection process. Second, a two-tier task structure in the community-based model helps achieve the delicate balance between exploitation and exploration in a distributed system. In studies of organizational learning, quality and innovation are typically seen as two antithetical objectives that juxtapose the refinement of an existing technology with the invention of a new one. It is generally assumed that there is a tradeoff between exploitation, the improvement of existing procedures, and exploration, the experimentation with new alternatives (Levitt & March, 1988; March, 1991). It is also argued that exploration of new alternatives reduces the speed at which skills related to existing procedures are improved; also, the improvements in competence at existing procedures make experimentation with others less attractive.

Indeed, in the firm-based model of knowledge creation, quality is an issue of production and innovation is one of product development. Nevertheless, our paper shows how under certain conditions product development can strike an excellent balance between exploitation and exploration to create a product of high reliability and innovativeness. The two-tier task structure, as we found, subsumes the tension between exploration (disorder) and exploitation (order), through which a large number of innovations are generated, selected, retained or rejected, and added to the production. Despite a dataset that spans only five years, our analysis demonstrates that the development community approximates an effective balance between exploration, as represented by the variations generated from the periphery, and exploitation, as represented by selection and retention by the core. It is true that finding an appropriate balance between exploitation and exploration is not a simple challenge, particularly because "the same issues occur at levels of a nested system – at the individual level, the organizational level, and the social system level" (March, 1991: 72). Organizational analysts for one do not observe the parallel code structure as often as linear structure or phased structure that hands off a project from a team to another in organizational settings. We see in this case, on the contrary, that when product development is embedded in evolutionary and cultural processes of learning anchored by criticism, the requirements of quality and those of innovation can be simultaneously satisfied across multiple levels.

Finally, the theoretical implications of the community-based model of knowledge creation extend beyond the literature of organizational learning that we have discussed so far. The alternative model of knowledge creation also changes the way organizational analysts understand community life and the uses of knowledge in society through virtual organizing and distributed work. The use of distributed knowledge for the purpose of innovation at the community level presents a unique model, in contrast with the firm or the market. Through differentiated norms and forms, community life and work become more structured and less chaotic, as individuals working on a distributed project take on different roles. Additionally, special structures sustain and enforce strong norms in the community, such that distributed projects can deliver products of high quality, although virtual communities often lack the tacit understanding that occur in face-to-face communications.

VI. FUTURE RESEARCH AGENDA

One of our intended contributions is to deduce a two-tier structure through our analysis of the developers' contributions documented in their emails to the Linux-kernel mailing list over a five-year period. Importantly, developers do not refer to each other as "the development team" or "the bug reporting team." For future research, how developer tasks and roles evolve is of critical importance to Open Source development activities. How did this task structure emerge and how and when and by what criteria do novice apprentices move up the rank in this structure? Specifically, to what extent are there pathways created by the organizers of Linux so that apprentice developers experience the decision process as one of their own making or are there more heavy-handed directions from more experienced participants? Moreover, how does the participation of highly motivated novices in Linux get actualized and then adjusted over time as these apprentices acquire skills? Do those who serve as developers do so immediately upon entering the Linux space or do they first act as novice users, from which a subset of developers may later emerge? Additionally, training through apprenticeship is one way to improve the skills of developers. Alternatively, discouraging and/or eliminating incompetent developers is another way to raise the overall skill level of the developers in the community. Successful migrations of trained volunteers through apprenticeship will indicate that the micro and the macro aspects of organizational life come together to create a functioning, productive, and sustainable organization.

As discussed in the methods and data section, the evolving skills of the apprentice developers are documented in "Credits" text files and "MAINTAINERS" text files over the years. These files constitute a public recognition of the people who have substantially contributed to the development of the Linux kernel and the nature of their contributions. What we need to understand is the process by which the information in these files is used to slot (channel) individual contributors into increasingly complex tasks as warranted by their contributions.

Although the case of Linux kernel development was useful in our construction of an alternative knowledge-creation model, we concur that this case has several limits. First, the Linux case may have limited generalizability because of the tasks involved. As shown in the case study, its product development process is based on a massive army, organized for mundane, labor-intensive tasks such as debugging, reviewing code, fixing code, and adding device drivers. So far, such a model has not been observed in cases where there is a disruptive innovation or technological breakthrough.

Second is the question of how applicable the Linux case is to other development projects.[18] The characteristics of Linux as a software product could make the case less generalizable, indeed. For example, the Linux operating system is a general-purpose tool that has numerous users who have the skills to participate in the development process or the willingness to invest in the training to learn how to develop such a tool. In addition, Linux is an open standard. As the number of user increases, network externality makes an open standard more valuable and subsequently increases the potential size of the development community. By contrast, products that target niche markets may not have a sufficiently large user base, which could potentially become developers. Even if some products do, their development projects are more difficult to sustain because they may not be able to provide sufficient returns of both economic and social benefits.

Furthermore, Linux is a digital good where the cost of production is significantly lower than that of conventional product development. For products that require heavy production cost and relatively little development cost, this model may not apply. Finally, the case of Linux could also be unique because computer source code is codified knowledge, which is explicitly documented in a text format. For development projects where the building blocks are tacit, it will be more difficult to design a knowledge-sharing mechanism. This suggests that an important area for future research is to examine the applicability of our community-based model by testing it with development projects for products that stand in contrast with the Linux kernel.

Despite the concern that the case of Linux can be regarded as rather unique, we provide some thoughts on how the model can be selectively adapted to apply in other settings. First, a commercial software firm can apply the community-based model of knowledge creation by combining open and closed models of software development. More specifically, 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 what we call "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.[19] It is important to recognize that without an infrastructure, 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

[18] We thank Paul Duguid for his helpful comments on the discussion of generality and externalities.
[19] 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 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

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.[20] For instance, the practice of knowledge sharing at the Netscape Communications Open Source project 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 (Lohr, 2000). This is a live example consistent with our argument that open-source licensing is a necessary condition, but not a sufficient condition, for the community-based model of knowledge creation.

The second area that the community-based model of knowledge creation can apply to is product development 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, geographically distributed employees and divisions of large multi-divisional business firms may adapt the community-based model and promote knowledge sharing within the boundary of the firm without losing their intellectual property rights to the public. Like the Linux developers, firms can also take advantage of the Internet technologies, which provide standardized communications protocols and connect different computing platforms across the globe, to promote collaboration.

[20] 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.

Nevertheless, 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. In summary, while the challenges are many, there may well be ways, such as work practice restructuring, to reap the benefits of the community-based model of knowledge creation.

APPENDIX A: The Inductive Method

The evidence guiding our process of theory development is divided into the following categories:

1. Materials related to the Open Source movement: We searched through numerous web sites to identify online white papers on the Open Source movement.
2. General information on Linux: The sources of information include The Linux Documentation Project, Linux Frequently Asked Questions (FAQ), Linux Links.com, and a search engine called Linux Knowledge Base. There is also a mailing list archive that tracks various development projects of Linux, address issues arising in a non-development environment, and discusses development and production topics. Using the World Wide Web, we also studied public documents related to Open Source and Linux. Some examples are OpenSource.Org,[21] Linux Frequently Asked Questions (FAQ),[22] Linux Knowledge Base search engine,[23] and The Linux Care Kernel Traffic newsletter.[24]
3. Technical information on the Linux operating system. Our main source of technical information on operating system is a college-level computer science textbook by Professor Andrew Tanenbuam (1987). The Linux source code is also an excellent piece of technical material. In addition, we collected many web sites that describe technical projects that are related to the Linux operating system, including an about.com expert information web site on Linux.
4. Information specific to Linux kernel. The Linux Care Kernel Traffic is a weekly electronic newsletter covering the activities of the kernel development mailing list.

[21] http://www.opensource.org
[22] http://www.tux.org/lkml/
[23] http://www.linuxcare.com/help-yourself/kbsearch/simple-search.epl
[24] http://kt.linuxcare.com/

5. The Linux kernel email archive (1991 to present).
6. The Linux-kernel survey (see Appendix B).
7. Transcripts of interviews with public figures in the Open Source movement. We gathered interview transcripts from Linux Focus, an online magazine, and First Monday, a peer-reviewed journal on the Internet.
8. Informal discussion with other people interested in the Open Source development model.

APPENDIX B: "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 their web site for anyone to download.

We estimate the response rate of the Linux-kernel survey to be 2.4 percent.[25] 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.[26]

[25] 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.

[26] In Table 3, 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.

APPENDIX C: An Example of Development Community Protocol

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 pieces is also copyrighted. The author of the scheduling program in the example below has accomplished an important task without having to rewrite a particular part of the code, 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
```

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.

REFERENCES

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

Becker, H. S. (1958), "Problems of Inference and Proof in Participant Observation," *American Sociological Review*, 23: 652-660.

Boisot, M.H. (1998), *Knowledge Assets: Securing Competitive Advantage in the Information Economy*, New York: Oxford University Press.

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.

Campbell, D.T. (1960), "Blind Variation and Selective Retention in Creative Thought As in Other Knowledge Processes," *Psychological Review*, 67: 380-400.

Castells, M. (2001), *The Internet Galaxy: Reflections on the Internet, Business, and Society*, New York, NY: Oxford University 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.

Cole, R.E. and G.K. Lee (2002), "The Linux Model of Software Quality Development and Improvement," Working Paper.

Cusumano, M.A. and R.W. Selby (1995), *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, New York: Free Press.

Davenport, T.H.

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

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

Davenport, T.H. and L. Prusak (1997), *Information Ecology: Mastering the Information and Knowledge Environment*, New York: Oxford University Press.

Deci, E.L. (1975), *Intrinsic Motivation*, New York: Plenum.

Dreyfus, H.L. (2001), *On the Internet (Thinking in Action)*, New York, NY: Routledge Press.

Eccles, R.G. and D.B. Crane (1988), *Doing Deals*, Boston: Harvard Business School.

Edmondson, A.C. (1996), “Learning From Mistakes Is Easier Said Than Done: Group and Organizational Influences on the Detection and Correction of Human Error," Journal of Applied Behavioral Science, 32(1): 5-28.

Festinger, L. (1961), "The Psychological Effects of Insufficient Rewards," *American of Management Review*, 4: 75-86.

Feyerabend, P.K. (1978) *Against Method: Outline of An Anarchistic Theory of Knowledge*, London: Verso.

Hayek, F.A.

(1945), "The Use of Knowledge in Society." *The American Economic Review*, 4: 519-530.

(1982), *Law, Legislation and Liberty*, V. I. Routledge & Kegan Paul, London

(1989), "The Pretense of Knowledge." *The American Economic Review*, 79: 3-7.

Hedberg, B.L.T., P.C. Nystrom and W.H. Starbuck (1976), “Camping on Seesaws: Prescriptions For A Self-designing Organization,” *Administrative Science Quarterly*, 21: 41-65

Hutchins, E. (1993), "Learning to Navigate," in S. Chaiklin and J. Lave (eds.), *Understanding Practice*. Cambridge University Press, Cambridge, U.K., pp.35-63

Kogut, B. (2000). “The Network As Knowledge: Generative Rules and The Emergence of Structure," *Strategic Management Journal*, 21: 405-425

Kogut, B. and U. Zander (1996). “What Firms Do? Coordination, Identity, and Learning," *Organization Science*, 7: 502-514

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.

Laczo, R.M. and K.A. Hanisch, (1999) “An Examination of Behavioral Families of Organizational Withdrawal in Volunteer Workers and Paid Employees,” *Human Resource Management Review*, 9(4): 453-477

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

Lawrence, P.R. and J.W. Lorsch, (1967), *Organization and Environment: Managing Differentiation and Integration*, Boston: Graduate School of Business Administration, Harvard University.

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.

MacDuffie, John Paul (1997), ”The Road to ‘Root Cause’ Problem-Solving at Three Auto Assembly Plants,” *Management Science, 43: 479-452.*

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

March, J. (1991), “Exploration and Exploitation in Organizational Learning,” *Organization Science*, 2(1): 71- 87

Markus, M.L., Manville, B. and Agres, C.E. (2000), “What Makes a Virtual Organization Work?” *Sloan Management Review*, pp. 13-26.

McGowan, D. (2001), “Legal Implications of Open-source Software,” University of Illinois Law Review, 1: 241-304.

Moon, J.Y. and Sproull, L. (2002) “Essence of Distributed Work: The Case of the Linux Kernel,” In P. Hinds and S. Kiesler (eds.) *Distributed Work*, Cambridge, MA: MIT Press, pp. 381-404.

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

O'Hara-Devereaux, M. and R. Johansen (1994), *Global Work: Bridging 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.

(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.

(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.

Pearce, J.L. (1983), "Job Attitude and Motivation Differences Between Volunteers and Employees From Comparable Organizations," *Journal of Applied Psychology*, 68(4): 646-652.

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

Popper, K.

(1972), *Objective Knowledge*, London: Oxford University Press.

(1973), "Evolutionary Epistemology," *Popper Selections*, in D. Miller (ed.), Princeton, N.J.: Princeton University Press, pp. 78-86

(1985), *Popper Selections*, in D. Miller (ed.), Princeton, N.J.: Princeton University Press

(1989), "The Critical Approach Versus the Mystique of Leadership," *Human Systems Management*, 8(4): 259-265.

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.

Schon, D.A.

(1983a), *The Reflective Practitioner*, New York: Basic Books.

(1983b), “Organizational Learning,” in G. Morgan (ed.), *Beyond Methods*, Beverly Hills, CA: Sage.

Simon, H. (1996), *The Sciences of the Artificial*, Cambridge, MA: MIT Press.

Slevin, J. (2000), *The Internet And Society*, Cambridge, UK: Polity Press

Staw, B.M. (1976), *Intrinsic and Extrinsic Motivation*, Morristown, N.J.: General Learning Press.

Tanenbaum, A. (1987), *Operating Systems: Design and Implementation*, Englewood Cliffs, NJ: Prentice Hall.

Teece, D.

(1986), "Profiting from Technological Innovation: Implications for Integration, Collaboration Licensing and Public Policy," *Research Policy*, 15 (December), pp. 285-305.

(1998), "Capturing Value from Knowledge Assets: The New Economy, Markets for Know-How, and Intangible Assets,” *California Management Review*, 40, 3, pp. 55-79

(2000), *Managing Intellectual Capital*, Oxford, UK: Oxford University Press.

Torvalds, L.

(1992), Emails sent by Torvalds in 1992 to the comp.os.minix newsgroup, archived by Linux International at http://www.li.org/linuxhistory.php

(1999), "The Linux Edge,"*Open Sources: Voices from the Open Source Revolution*, DiBona, Ockman, & Stone (eds.), Sebastopol, CA: O'Reilly & Associates.

Tsoukas, H.

(1994), "Introduction: From Social Engineering to Reflective Action in Organizational Behavior," in H. Tsoukas (ed.), *New Thinking in Organizational Behavior*. Butterworth-Heinemann, Oxford, pp. 1-22.

(1996), "The Firm As A Distributed Knowledge System: A Constructionist Approach," *Strategic Management Journal*, 17: 11-25

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.

Van de Ven, A.H. and M.S. Poole (1995), “Explaining Development and Change in Organizations,” Academy of Management Review, 20 (3): 510-540.

Weick, K.E. (1977), "Organizations as Self-Designing Systems," *Organizational Dynamics*, 6(2): 30-46.

Weick, K.E. and L. Berlinger (1989), "Career Improvisation in Self-designing Organizations," in M.B. Arthur, D. Hall, and B.S. Lawrence (eds.), *Handbook of Career Theory*, New York: Cambridge University Press.

Weick, K. and F. Westley (1996), "Organizational learning: Affirming an Oxymoron," in Stewart Clegg, Cynthia hard and Walter Nord (eds.) *Handbook of Organizational Studies*, Thousand Oaks, CA: Sage Publications.

Wessel, D. (2001). "Decentralization and Downtowns," *Wall St. Journal*, (Oct 25): p.1.

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

Znaniecki, F. (1934). *The Method of Sociology*, New York: Farrar & Rinehart.

TABLES

Table 1. . The Community-based Model vs. the Firm-based Model of Knowledge Creation

Organizing Principles	The Firm-based Model	The Community-based Model
(1) Intellectual Property Ownership	Knowledge is private and owned by the firm.	Knowledge is public, but can be owned by members who contribute it as long as they share it.
(2) Membership Restriction	Membership is based on selection, so the size of the firm is constrained by the number of employees hired.	Membership is open, so scale of the community is not constrained.
(3) Authority and Incentives	Members of the firm are employees who receive salaries in exchange for their work.	Members of the community are volunteers who do not receive salaries in exchange for their work.
(4) Knowledge Distribution Across Organizational and Geographical Boundaries	Distribution is limited by the boundary of the firm.	Distribution extends beyond the boundary of the firm.
(5) Dominant Mode of Communications	Face-to-face interaction is the dominant mode of communication.	Technology-mediated interaction is the dominant mode of communication.

Table 2-1. 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 2-2. A Chronology of Experimental 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 experimental releases have odd version numbers (e.g., 2.1, 2.3). Users who rely upon a Linux operating system to support their production computing typically prefer the stable releases, in which new releases introduce only well-tested new functionality, no new bugs, and no backward compatibility problems. By contrast, users who want to try out new ideas and get feedback on them as rapidly as possible prefer the experimental releases.

Table 3. A Two-tier Structure With Four Categories of Developers, 1995-2000

Emergent Roles of Linux Developers	Number of people	Total Number of Emails [NOTE 4] Sent to the Mailing List	% of Total Emails Sent
Core			
Project Leader	1	2,840	1.4%
Maintainers	121 [NOTE 1]	37,387	18.8%
Periphery			
"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.

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. 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.[27] Their tasks include identifying bugs, characterizing bugs, and eliminating bugs. We have also found some overlap between the development team and the bug reporting team. Forty-nine percent of the bug reporting team has 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.

[27] "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.