Distributed Knowledge and the Global Organization of Software Development

Anca Metiu*
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104-6370
(215) 898-4191
metiu@management.wharton.upenn.edu

Bruce Kogut**
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104-6370
(215) 898-1093
kogut@wharton.upenn.edu

February 2001

We would like to thank CIBER at the Anderson School, UCLA, the Wharton-SMU Research Center at Singapore Management University, Carnegie Bosch Institute, the Reginald H. Jones Center at the Wharton School for financial support of this project, and the many managers and engineers who gave us their time.

* Wharton School, University of Pennsylvania.
** Wharton School, University of Pennsylvania and Centre de Recherche en Gestion, Ecole Polytechnique.

Abstract

The growth of a global infrastructure has enabled the spatial dispersion of work activities. The software industry permits the study of the potential and limits of new ways of organizing work over borders to exploit globally the opportunities opened by the digitalization of production and products. Based on field observations of companies in four countries, we identified two distinctive models of organizing global software development. "The global project model" currently dominates the organization of global outsourcing activities in software development. In this model, multinational corporations seek to utilize cost differentials between locations and to bring products to the market more speedily by exploiting time-zone differences and the dispersed global capabilities. "The open development model" is destined to allow talented professionals around the globe to use their competences to the maximum. While not necessarily organized by multinational corporations, the open development model nonetheless affects their activities.

Introduction

One of the hidden lessons from the many studies on international business is that trade and investment just do not happen. They are activities organized and managed by firms and often by multinational corporations. The multinational corporation exists because it is able to carry out trade and investment at lower cost than alternatives and because it is able to exploit better the differential capabilities of nations.

Theories of the multinational enterprise argue that the existence of these forms of organization is due to their efficient transfer of firm-specific knowledge. McManus (1972) offered one form of this argument by explaining that because firm-specific knowledge is a public good that can be transferred at zero marginal cost, it is difficult to appropriate the returns from its use. The role of multinational firms is precisely to exploit and appropriate these returns (Buckley and Casson, 1976). As Hennart (1982) elaborated, multinational firms organize exchanges more efficiently than markets because they economize on information channels, their managerial fiat leads to quick decisions, and obedience to directives leads to unified action. More recently, Kogut and Zander (1993) endorsed this view that the multinational corporation exists as an efficient means for transferring knowledge, but not *necessarily* due to resolution of transaction cost as so much as a superior vehicle for disseminating tacit and organizational knowledge.

Though multinational corporations are to be found in almost all countries, their activities are heavily concentrated among wealthy countries. In 1998, 71% of foreign direct investment went to ten developed countries (United Nations, 2000). It is easy to see why from the simple economics of location and trade. Economic models explain the geographic concentration of economic activity by the location of demand, by factor and transportation costs, and by

economies of scale in production (Krugman, 1991; Buckley and Casson, 1976). These approaches imply that, for non-negligible transportation costs, production will often be located near sites of demand and first-mover advantages, that is, in rich countries.

Software development represents, in many ways, a limiting case for these models and the economics suggest a tremendous advantage of offshore production. Software is digital and it is transported by satellite transmission. Thus, transportation and communication costs are insignificant. Moreover, scale economies are low. There is a large pool of highly skilled professionals in many places, some working at markedly lower labor costs than what we see in the developed markets such as the US. In 1997, while an entry level software programmer recruited by one of India's top domestic software companies could expect to earn about $500 a month, his American counterpart earned approximately $3,000 or much more (*Financial Times* Dec. 2, 1998).

Coupled with the institutions of technical education and with advances in infrastructure development, these economics have made India the most prominent example of the emergence of a new industrial center. In India, this young industry (the first software companies were established in the mid-1980's) has been growing at annual rates of 50-55% over the past eight years. Export earnings, estimated at $2.6 Billion in 1999, are also growing rapidly. Out of this volume 59% of software exports are to the US (Nasscom-McKinsey Report, 1999). An Indian National Task Force on Information Technology recently set a target of $50 Billion exports by 2008, when it predicts software will be India's biggest export (*Financial Times*, March 15, 1999).

The predictions of economic models of location are challenged by two important observations. The first is that most software development, especially for innovative products, is

located still in wealthy countries. In 1999, $80.9 Billion of the $150 Billion global software industry revenues accrued to the US (US Census Bureau, 1999).

The second observation refers to the deficiency of organizational foundations to support the coordination of skilled labor. As seen also in the currency trading rooms studied by Zaheer (1999) that also exchange "bits" over distances, one form of this new organization exploits the efficiency of 24-hour projects. Yet, even in the absence of conducting around-the-clock projects – which appears to be the rare case in software development – firms have invented new organizing techniques to exploit more efficiently costs and capabilities on a global scale.

In the following, we report on field observations regarding two distinct models of organizing global software development. The first model, which we label "the global project model", currently dominates the organization of global outsourcing activities in software development. In this model, multinational corporations seek to utilize cost differentials between locations and to bring products to the market more speedily by exploiting time-zone differences and the dispersed global capabilities. The second model, which we label "the open development model", is destined to allow talented professionals around the globe to use their competences to the maximum. While not necessarily organized by multinational corporations, this model nonetheless affects their activities.

There is a wider implication in our analysis beyond these two models. It has long been a puzzle why activities have a regional characteristic. Cantwell (1989) re-initiated this line of study by positing that direct investment flows to countries and regions where there is a technological advantage. What happens dynamically where there are major cost differences in labor costs, when ideas and people flow back and forth among countries, and regions evolve in their capabilities? The persistence of regional differences is no longer, as argued above, based

on transportation and communication costs and first mover advantages on scale economies. We conclude, therefore, on a note that speculates on the evolution of regions such as Bangalore as major centers of global activity.

Previous studies of software development

Studies of software development have focused heavily on its iterative and creative character, which make issues of location and proximity extremely salient. Wang, Barron and Seidmann (1997) argue that the make-buy decision is influenced by informational disparities inherent in software contracting. They show that contracting and the allocation of work are influenced by the specific industry know-how developed by outsourcers. Empirical studies support the claim that an internal developer has a substantial advantage over an outsourcer (Nelson, Richmond and Seidmann, 1996; Lacity, Willcocks, and Feeny, 1996) and recommend that organizations develop internally projects that require familiarity with existing business processes.

The tension between creative and well-understood tasks is central to any understanding of the location of software development. Cusumano (1991) documented how software design moved from art to routinized tasks manipulating standardized modules. His study showed the rise of software "factories" first in the United States and then spread to Japanese companies such as Hitachi, Toshiba, NEC, and Fujitsu. In the effort to make the development of software more cost-effective, firms proceeded to standardize software tools and environments and to re-use code in different projects. This approach culminated in an attempt to rationalize the entire cycle of software production, installation, and maintenance through the establishment of factory-like procedures and processes. Cusumano concludes that software factories were not, however,

appropriate to new and complex systems because the processes involved in this type of projects are only imperfectly understood.

Both quantitative and qualitative studies found that problem solving and creative processes pose important challenges to the organization of software development. In one of the few surveys of software development projects, Kraut and Streeter (1995) found that their coordination will be difficult when specifications are incomplete, and when the knowledge needed is not documented and therefore not easily available. They also found that coordination was impeded as project size and complexity increase, and that the maintenance of informal communication networks was an important determinant of project success.

Ethnographic studies of software development uncovered the importance of co-presence and proximity for this activity. For example, Perlow's (1997, 1999) investigations of software developers' use of time revealed that software development work involves both individual cognitive activities, which require long periods of uninterrupted time during which one can concentrate, and collaborative activities, which require periods of interaction with others. In one of the rare ethnographic studies coordination among programmers across geographic space, O'Riain (1999) concludes that close, face-to-face communication is needed for conveying complex information and for building and sustaining trust.

Several other studies videotaped and examined software writing (Adelson and Soloway, 1985) and design meetings in co-located (Tang, 1991; Walz, Elam and Curtis, 1993) and dispersed settings (Olson and Bly, 1991). They found that successful problem solving demands that the team achieves a common understanding of the work, as well as a strong sense of the identities of the people involved in the project. All these objectives are better achieved in the rich medium of face-to-face interaction. While recent advances in technology can better support

such interaction than before, these studies again suggest the difficulty of coordinating software projects over distance.

The above studies pose, but do not resolve, the question whether innovative tasks can be coordinated over space. It is useful to step away from these studies and to identify more generally their implied impediments. Before turning to the field evidence, we first establish three factors that influence the costs of allocating software development to different sites.

Drivers of Sourcing Costs

In our view, the globalization of software depends upon coordination of modularized tasks, communication, and social context.

Coordination

Coordination in software relies heavily on the use of modular design principles. The original craft orientation of software development was badly strained by the demand for large software systems that required 1000s of engineers. Frederick Brooks' book *The Mythical Man-Month* (1975) documented the challenges posed by the creation of the operating system for the IBM 360 large frame computer. This experience led Brooks to formulate what has come to be known as Brooks' Law: adding personnel was not a solution to solving a bottleneck or increasing speed because of the costs of training new members and of communications overhead. Recently, McConnell (1999) argued that Brooks' Law holds only insofar as managers mis-estimate the time needed for the completion of projects that lack reliable estimation, tracking, and training. However, controlled projects are able to add staff later in the project because of their better documentation and their task partitioning.

More productive sources of improvement were sought in organizing and writing code better. Software languages improved dramatically since the 1960s, when the introduction of Unix, Lisp, and C supported the decomposing of development work into small modules that permitted teams to work with minimal communication. Modularity, however, leads to a clear hierarchy that puts a lot of strain on architectural design (Cusumano, 1991). The methodologies to aid this structure have evolved from fairly simple waterfall models to more sophisticated systems such as the iterative model and the spiral model (in essence an iterative model done over short time cycles).

The coordination of software projects retains some common elements across various models of software development. User needs and/or requirements are defined and the architecture is created. These activities are often done iteratively with the customer or with the in-house users. Co-located or distributed teams then carry out detailed design and debugging . There are a variety of different models, as discussed below, concerning the location of the teams and how often they communicate. The products are sometimes prototyped for customer approval. More standard is the use of gap analysis to test the satisfaction of the customer requirements by the achieved performance of the software product.

Modularity is fundamental to ability to implement global sourcing. In their study of computer production, Baldwin and Clark (2000) show how modularity is the mechanism by which sourcing from the market is built into the design of the product. By sourcing externally, the assembler permits the supplier to take on more responsibility for the innovation in the module itself. Therefore, modularity permits an efficient exploitation of the differential abilities of various locations by allowing the different work tasks of a software project to be assigned to

the most efficient producer. Traditionally, routine software modules were assigned to low cost sites.

However, coordination across borders is especially difficult when the task is creative. Whether defined as adaptive adjustments or as radical departures from established practices (Perrow, 1986), creative projects pose a challenge to the coordination of dispersed teams. Because such tasks cannot be broken down into small and well-defined components, and because they often require variation in the needed knowledge bases, creative tasks are likely to pose problems to distant coordination.

Communication Media

Spatially distributed sourcing of software could not be possible without the tremendous drop in communication costs and the increase in bandwidth over the past decade. For places like Bangalore or Hyderabad in India, satellite communication to the United States is easier than intra-Indian communication using the national network. The increase in bandwidth has also enabled the use of video communication, in addition to voice and electronic messaging. As Orlikowski and Yates (1994) have shown, electronic media enable a rich and varied array of communicative practices. Not only are users of electronic mail able send memos to each other; they also have the ability to dialogue with one another because the medium permits them to embed parts of previous messages in new messages.

The social psychology literature has studied the types of tasks for which co-location and face-to-face interactions are important. This literature holds that face-to-face communication is needed when the task is ambiguous (Daft and Lengel, 1986) and for substantial consensus decisions (McKenney, Zack and Doherty, 1992). McKenney et al.'s (1992) study of the

communication patterns in a programming team finds that electronic mail and face-to-face are complementary media: while email provides efficiency in well-defined contexts, face-to-face has the ability to build a shared understanding and definition of the task. Coordination at a distance is costly if not impossible because the medium of communication (mainly the computer) misses a whole host of non-verbal cues that influence message reception and understanding.

Researchers have long made a powerful case for the need for face-to-face communication. The psychologist Paul Ekman (1985) notes, for example, that face-to-face communication provides many cues for truthfulness. Nohria and Eccles (1992) argue that face-to-face communication has three main advantages over electronic communication: the existence of a shared context among participants, its richness which encompasses the full bandwidth of physical senses and psycho-emotional reactions, and its capacity for interruption, repair, feedback and learning.

In a particularly important study, McGrath and Hollingshead (1994) discuss in detail the limitations of computer-mediated communication. They observe that this medium often alters communication times, as well as the sequence and synchrony of messages. At the same time, it increases ambiguity because the sender cannot be confident that failure of any given member to reply to a given message in a timely fashion reflects that member's deliberate choice (1994 p. 21). Groups using computer-mediated communication try to supplement the lack of nonverbal cues by using "emoticons" as well as a) longer and more complex syntax; b) jargon, argot, and other shared nonstandard language that relies on culturally shared connotative meanings; c) punctuation and format conventions; and d) redundancy (McGrath and Hollingshead 1994:19).

The difficulty of transmitting some kinds of knowledge required for software production over digital media is an important element to our understanding of why there are lasting

differences between the tasks performed in different regions. Clearly, the limitations imposed by the creative character of work tasks can be offset, in part, by modularity. However, as we discussed above, some tasks cannot be modularized; or, if a certain degree of modularization can be achieved, the task still requires large overlaps between various knowledge bases. In such cases, rich and intense communication is needed for the sharing of competencies. In the words of Frederick Brooks, "since software construction is inherently a systems effort - an exercise in complex interrelationships - communication effort is great, and it quickly dominates the decrease in individual task time brought about by partitioning." (1975:19).

Social Context

The modularity of design and improvements in communication are powerful forces for distributed software development. However, there is a limiting factor that makes coordination over distance difficult. People interpret the world in the different contexts in which they live.

We start from the premise that for creative tasks, sustained involvement is needed. Next, people's spontaneous involvement in the task is not only detectable by team members, but also becomes 'contagious' for group members. Such sustained and spontaneous involvement is a characteristic of face-to-face interactions.

This type of interaction creates an environment of common understanding, mutual respect and emotional closeness that supports the free expression and discussion of ideas, and hence for creativity. As Goffman (1961) writes, in face-to-face encounters, the fact that activity "is going on before one's eyes ensures that a mere definition of the situation is experienced as having the thickness of reality. That other persons are involved ensures that engrossment must be steadily sustained in spite of the flickering of one's actual interest" (1961: 41). Conversely, physical

distance allows for periods of lack of interest and for wide variation in attitude and feelings (Goffman 1961: 41).

This human capacity to express and to recognize emotions is key to understanding how co-presence and proximity support an atmosphere of sustained creativity. Lave and Wenger (1991) show that learning some crafts requires not simply a transmission of abstract concepts, but also effective co-participation in an activity. Such concrete actions, performed together, give access to and refine a wealth of nonverbal communication cues that can enhance understanding and speed execution. More importantly, the knowledge being created in face-to-face interactions entails often the assessment of others' degree of engrossment in the common activity. Such engrossment with the task and engagement with co-workers for successful task completion is hard to detect across physical distance (Metiu, 2000).

For some kinds of activities, this does not pose a problem. For example, many software products include sophisticated mathematical algorithms used in data compression for telecommunications. Because intelligence and mathematical ability are far more fairly distributed over the earth's surface than economic production and wealth, it is a common policy that software products using sophisticated math go to where the mathematicians are.

Just like math skills, engineering competence is also distributed in the world. But engineering skills are often also acquired through practical experience and exposure to customer problems. Asking engineers to write code for well-defined customer requirements is feasible to implement globally. However, it is hard for the customer, even when it is a manager of the division of the same firm, to co-create with an engineering team located far from the user market. As von Hippel (1994) noted in his studies on the locus of innovation, suppliers have difficulty innovating to meet new customer requirements when far from the market.

Close interaction with the customer is particularly important when the requirements are unclear, or are changing during the course of the project. When the software developers are unfamiliar with the functionality of the end product of their work, the communication of the product's features may be difficult. The shared context that is the background to innovative work is not embodied in bits of information digitally communicated. This context exists prior to communication, and increased bandwidth will not easily resolve this obstacle to global innovation. Therefore, the impact of a shared context is particularly important in the case of creative tasks.

Two Models

Historically, the dominant strategy for offshore sourcing has principally been to lower costs and to expand the labor pool of software engineers. For decades, the demand for software engineers has grown steadily. Whether this increasing demand represents a labor shortage (ITAA, 1997) or a simple market tightening (Barnow, Trutko, and Lerman, 1998) is still under debate. Using Bureau of Labor Statistics data, Barnow et al. (1998) concluded that over the period 1988 to 1997, employment in the IT (information technology) occupations grew by 64%. Within IT jobs, the category 'Computer Systems Analysts and Scientists' has grown the fastest, by 158%.

The expansion and growth of offshore development sites was also the result of tightening US immigration laws that have discouraged the policy of importing labor even on short-term contracts. For many companies, global sourcing was also a way to resolve the Year 2000 problem that required checking the code of the older programs and to maintain the legacy

software systems (legacy software is software written in a language, such as Cobol, that is no longer used for new development.)

The creation of offshore sites, either by establishing in-house subsidiaries overseas or by contracting to offshore suppliers, provided the talent to accomplish these tasks. Since the offshore sites also were anxious to prove their competence by following ISO standards (Arora and Asundi, 1999) or by achieving quality rankings such as those of the Software Engineering Institute at Carnegie Mellon University, they tended to document more and better. Therefore firms not only had access to competent labor, they often had better documentation and quality standards.

Offshore sites also led to the recognition that product cycle times could be improved. The efforts to lower costs and to speed up development time express a fundamental tradeoff. Speed can be gained, normally, by increasing costs. Global sourcing reduces this cost by utilizing people across time zones, often by tapping into lower wage rates.

Continuous development whereby global teams hand off their work at the end of their workday to members arriving at work in a westerly direction is still a model to speed up development by managing the productivity of the least productive units. It assumes that the product innovation is already accomplished, usually by the site in the developed country.

Global sourcing rarely touches the innovation process itself. It relies upon the global project model of finding the best way to get speed at the right cost. The implicit model of innovation is that good ideas come from innovators usually located near customer needs. These innovators create the architecture and the design. The open development model of getting more innovation on a global basis, while rarely an objective of current global sourcing policies, presents an important alternative to traditional models. Before we present evidence in support of

this claim, we need to understand better the global project model, which is currently the dominant model for the global outsourcing activities in software development.

Data Sources

Our data consist of interviews and case studies. Over the course of two and a half years, the authors traveled to companies in four countries (the US, India, Ireland, and Singapore) to conduct interviews with managers of software development projects and with engineers involved in dispersed software work. We conducted extensive semi-structured, in-person interviews with 80 informants (managers and engineers) in 18 companies. Because our goal was to generate a theoretical sample (Glaser and Strauss, 1967), we sought to maximize the diversity of our informants along type of organization (firm employees or voluntary contributors), form of organization (wholly-owned subsidiary of multinational corporation or foreign firm), country represented, and job-related characteristics (managers or engineers).

We interviewed 13 managers in five US companies, and 30 managers in 9 Indian companies. In Ireland, we interviewed 25 managers and engineers in two companies. Finally, in Singapore we interviewed seven managers in two companies. We also interviewed five developers active in the open source movement

Our goal to obtain an in-depth understanding of the issues involved in dispersed software development prompted us to follow some of Miles and Huberman's (1984) suggestions. As such, we tried not to lead any of our interviewees' responses, and not to share our knowledge or our guesses with them. More importantly, one author spent one week with an Irish company and two weeks with an Indian company to interview managers and engineers, observe people as they

went about their work, and write detailed case studies on these companies' software development models.

The interviews and observations were structured around three primary foci: 1) the impact of information technologies on the organization of software development work; 2) the problems that confronted managers and engineers, and 3) the solutions with which they were experimenting. As data collection advanced, we revised our framework such as to eliminate repetitive questions and to focus on issues that emerged as relevant.

We analyzed our data by following the constant comparative method recommended by Glaser and Strauss (1967). The aim of this method is to generate rich descriptions of phenomena that allow the researchers to generate theories about them. We identified theoretical categories and refined them by making comparisons across categories. During data analysis, we kept an eye for disconfirming evidence (Miles and Huberman, 1984) in order to ensure the validity of the categories we developed. This approach enabled us to arrive at a conceptualization of several models for the organization of work in global software development.

The Global Project Model

The attractiveness of offshore development prompted firms to develop strategies to overcome the obstacles to virtual development of software.

The most modest of these strategies is to simply exploit the cost advantage by assigning routine tasks to offshore sites. This strategy starts with the concept of the software factory (Cusumano, 1991) – but played out across space. These tasks might be the maintenance of legacy systems, developing highly specialized but routine modules, or completing entirely the detailed design work of a software project. In such tasks, modularity is easy to achieve,

communication requirements are low, and there is little concern over shared context with the end user. As shown in Figure 1 for the traditional waterfall model, module tasks are separable and, thus, easy to locate in different places.

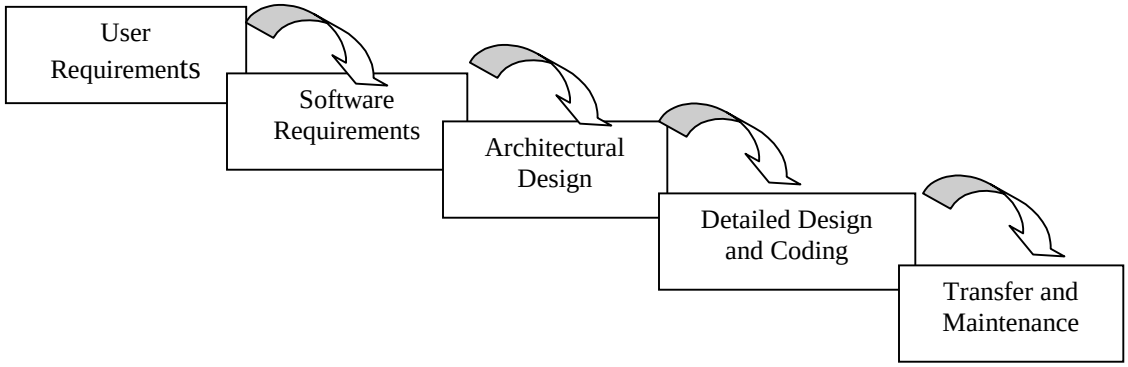
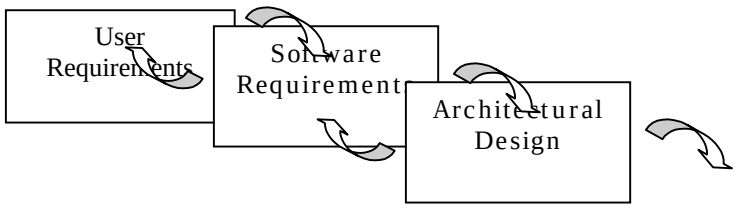


Fig. 1. The waterfall model minimizes the need for coordination and communication because it sees the process of software development as a cascade of phases, the output of one being the input to the next.

Many American firms employ this strategy by sub-contracting extensively to Indian software houses. These houses, such as Tata Consultancy Services, WIPRO, Infosys, and others vary in their services and strategies. A common feature, though, is their offshore development centers. These consist of silos – people and facilities – dedicated to a firm, be it Japanese, European, or American, which act as if they were a unit inside the customer's organization. To protect the intellectual property that may be involved, the engineers assigned to different silos

However, some customer companies are seeking active ways to speed up problem-solving in innovative projects. Unlike the earlier model in which software development follows a sequence of steps, the goal to decrease innovation time relies upon a more iterative process, as shown in Figure 2. Often though, the higher communication and coordination needs of iterative projects exceed the cost advantages of offshore sites.



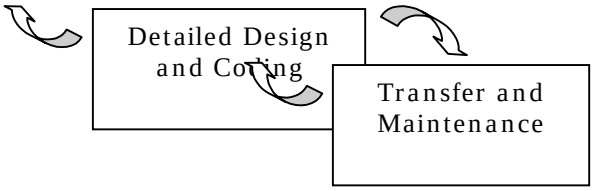


Fig. 2. The iterative model recognizes the substantial overlap between various phases of software development. For example, the architecture stage also requires some coding, while coding itself may require changes in design. Also, some stages such as user requirements and maintenance require close interactions with the customer.

One of the firms we studied is a large US telecommunications company that faces the strategic problem that customers are demanding it provide feature sets embedded in software rather than in hardware. This request causes a problem of enormous importance for the company. The new emphasis on software leads them to move to a faster development model. Because they are hesitant to commit to outsourcing for strategic products, they dramatically scale up their own in-house software operation at overseas sites, while identifying a few long-term partners for outsourcing. While still uncertain about the direction of this development model, the company addresses the new challenges by intensifying communication and travel between the

Another firm confronts a similar situation. It is a large US consumer electronics company. Unlike the telecommunications company that sources its compression algorithms from its own sites around the world. But its new Internet businesses are demanding software that is more sensitive to customer needs. The manager in charge of offshore development, confident in the remote sites' abilities, competes with other firm units to sell its services to product divisions. Yet the company is worried about offshore development's effect on its domestic employment levels, and it has yet to diffuse a standardized process for rapid product development using dispersed sites. These two cases show that the growth in the demand for software has yet to be reflected in the multinational corporations' top-level strategies.

Offshore Comes Onshore

While the large customer companies move rapidly to increase offshore sourcing without clear product development models, many of the offshore houses are pursuing aggressive and targeted strategies to expand their operations and upgrade their capabilities. They are coming to "onshore." These companies want to be more than low cost vendors but to take part in innovation. The example of Infosys and WIPRO, two of the most successful Indian software companies, is telling. They aim to move from being a provider of software development services to US firms to being participants in global innovation by developing their own products. Ranjit Singh, who is responsible for Xerox's Internet software development, advanced an explanation of this phenomenon when he said that the limits of the "arbitrage" model that exploits lower costs offshore are being reached.[1] The very success of global outsourcing causes not only labor rates to converge over time to the similar levels around the world; it also causes global capabilities and aspirations to converge.

For years, many of the Indian companies have been sending engineers to work at their US clients' sites principally as a way to circumvent the immigration hassles of locating foreign engineers at temporary sites. The physical movement of engineers – or "body-shopping" – does not fulfill the promise of virtual software teams. However, stationing engineers in the high-tech centers of customer demand suddenly allows the offshore company to solve the problem of shared context with the end user, as the are involved in defining customer user needs. The Indian operations still do the detail design work, but increasingly the architecture and application functions are under their control.

[1] Comments made at the workshop on global software development, Reginald H. Jones Center, Wharton School, December 1999.

The question of interest becomes whether these developments lead to a blurring of the distinction between onshore and offshore. Can the locus of innovation be shared between the two sites?

For some companies, the solution is continuous around-the-clock development. Trintech is an Irish software company that produces credit card swiping equipment software. To support its strategy to move into innovative software products, it moved closer to the customer in Silicon Valley. However, this decision by itself puts this Irish company into the same dilemma as an American firm relying upon offshore sites for the development of some modules. Trintech's strategy is to operate 16-hour development cycles.

The company develops software in four centers, each with a specific expertise. Dublin specializes in electronic point of sale systems and e-payment and is the headquarters for physical world hardware and software development. San Jose, California, has built an expertise in e-commerce payment solutions. Princeton's highly skilled developers are strong in the mathematics of encryption and authentication. The Frankfurt center is responsible for the smartcard and chip technology and is the liaison to one of the most important financial capitals of mainland Europe. Though organized globally by competence, these dispersed operations also allow Trintech to perform software development work 16 out of 24 hours every day. At the end of the workday, work done in one center is virtually "handed off" to another center. The day starts with the Dublin engineers cutting the code that they subsequently ship both to Princeton and to San Jose. The code is completed and debugged at these centers, and then shipped back to Dublin before the Irish staff arrives to start their day.

The implementation of this strategy shows the arduous management that is required for virtual innovation. Process and product compatibility is achieved via thorough documentation of

engineering problems and solutions. Each developer's code is debugged by the other centers, and then ranked in terms of the number of bugs it contained. Gap analysis – the measurement of software performance against customer requirements – is strenuously imposed on development stages. Such Tayloristic controls and detailed specifications are needed to supplement the lack of informal interactions as a problem-solving mechanism. The time zone differences between locations create an important barrier to communication, as they offer only a narrow window for people to discuss issues with someone situated across the ocean. While development work often necessitates immediate feedback, sorting out a crisis between Dublin and San Jose – where there is only a 1-2 hour window – can take two days instead of just several hours.

This model represents an important departure from the early days of the software factory or the IBM 360 operating system. With the coordination mechanism provided by the Net, innovation becomes faster and cheaper. However, this model does not clearly enhance innovation itself: ideas remain the domain of the chief architect and the innovating team. This model exploits skilled labor globally, but it represents only a partial re-location of innovation to regions in emerging markets: originated in Dublin, Trintech's innovations are still done at this site.

The Open Development Model

Continuous diurnal development leads to a pronounced dispersion of activities across space because it lowers the costs of global sourcing. When used in practice though, the model still keeps control within a co-located group situated in a wealthy country. Open software development represents a radically different model of participation in innovation.

Whereas most of the better known examples of open software development are located primarily in wealthier countries, it is instructive to note that this model was observed during the field research in developing countries as well. As the offshore markets become more sophisticated, their ability to understand the needs of customers in developed countries will continue to increase. Sometimes the solution comes from the fact that the market itself comes to offshore. Sridhar Mitta of WIPRO tells the intriguing story of a final customer located in San Diego who requested development of a software product. Customer requirements were defined, the architecture drawn up, and the final product was shipped back to the customer. Mitta never met the customer.

This example demonstrates that innovation need not be located in the richest markets. Apart from the lower labor costs, coordination and communication costs were minimal because there was no subcontracting. For this product, the shared context was not consequential, and the Indian engineers were able to define user requirements.

Such examples happen, in reverse, more than we realized at the start of our investigation. Consider the example of the PalmPilot of 3 Com. This personal digital assistant relies on a proprietary operating system. However, 3 Com encourages complementary development of games and utilities that can be downloaded and installed in the handset. It provides tutorials, releases development tools through the Net, and helps distribute the freeware and shareware products. For "creators" who want to have their software stamped for quality, they can apply for platinum certification administered by a third-party provider. Some 3500 people are estimated to be engaged in programming for PalmPilot.

This is not an isolated example. Microsoft and Apple have both significantly increased their support of development tools for their operating systems. Compaq released software early

through the net to allow developers to test the performance of their software add-ons. Some companies have even made proprietary code available for communal development by releasing their operating system code. In 1998, Netscape released the operating system code for their Communicator 5.0 (Yoffie and Cusumano, 1999).

In order to understand the issues and possibilities involved in the open development model, we have to examine more closely the most prominent example of software development by spatially dispersed contributors: Linux.

Linux

Linux is a Unix operating system that was developed by Linus Torvalds and a loosely-knit community of hackers across the Internet. In 1991, Linus Torvalds, a Finnish Computer Science student, wrote the first version of a Unix kernel for his own use. Instead of securing property-rights to his invention, he posted the code on the Internet with a request to other programmers to help upgrade it into a working system. The response was overwhelming. What began as a student's pet project rapidly developed into a non-trivial operating system. Torvalds integrated the utilities already existing in the Unix public domain into his kernel and named the complete operating system after himself, -- Linux, for Linus' Unix.

Thousands of people around the world contribute to Linux. These hackers wrote the code for fixing bugs, utility improvements, and new features. They were responsible for writing the interfaces to the kernel. As all code is open for inspection and for change by anybody who can improve it, bugs are rapidly detected, and better code is in continuous development.

The coherence of the Linux operating system is due to the availability of source code and development tools, as well to the common culture shared by hackers. New code is submitted to

Torvalds, who decides whether or not to accept it, or request modifications before adding it to the Linux kernel. While Torvalds has ultimate authority over what goes into the kernel, several developers – such as the networking chief or the driver chief – have more-or-less control over their particular subsections. For software that does not go into the kernel, Linus Torvalds cannot prevent others from adding specialized features. Developers provide their own patches to the kernel, available separately. These patches allow even greater customization without risking support the integrity of the operating system for the vast majority.

The engineers who helped the development of Linux can come from anywhere that has web access. The development of Linux did not incur the costs of global sourcing. There were no fixed costs to be added for development, and there were minimal coordination costs. Moreover, the risk that a critical module may not be developed is low: since the users are also developers, if they need a module, they'll develop it. As Figure 3 shows, this model exploits the maximum of skills in the system. This is no longer the management of getting the least efficient link to complete its task in order to hand off the module to the integrator.

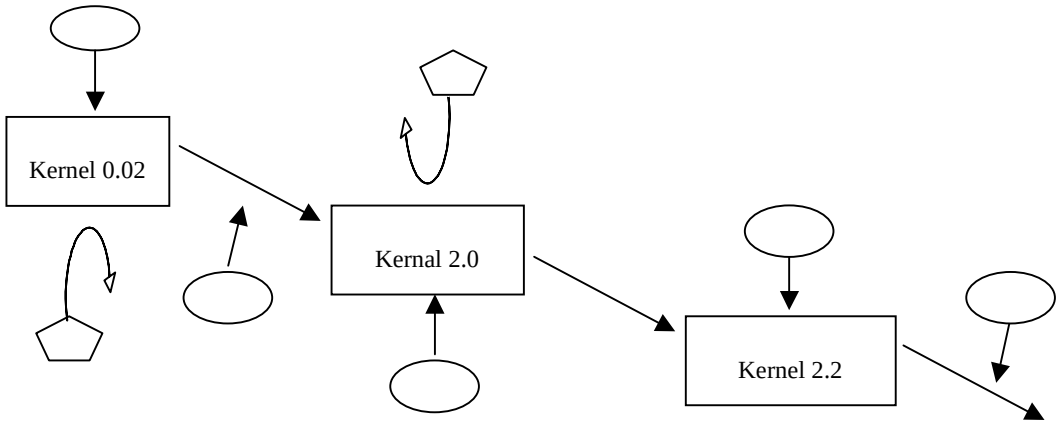


Fig. 3. The community model of open development. The sharing of source code and frequent releases increase the speed of debugging and hence the quality of new releases. Linux' modular structure allows developers around the world to contribute to its improvement. Not all new features become part of the

kernel, but they are still available for use and further development. The contributions of the Linux community members not only improve the kernel, but also pull it in an innovative direction.

Just like the PalmPilot case, a community of software engineers work to create an innovation, except this time the product is a radical innovation. Today, Linux has a 30% share of Internet server sites in 1999 (Dempsey, Weiss, Jones and Greenberg, 1999). However, with the recent initial public offerings of companies that compete on Linux distribution and support, commercialization threatens to divide the development community. Recently, the inability of some Linux contributors to gather the cash to buy the stock reserved for them in the recent IPO of Red Hat created discontent among developers. And their contribution to Red Hat is remarkable.[2] For a Linux product such as that sold by Red Hat, approximately 500 megabytes of the total 573 megabytes of compressed code have been developed by independent third parties, including approximately 10 megabytes of code contained in the Linux kernel. Included within the 573 megabytes of code are approximately 645 distinct software components developed by thousands of individual programmers.

Individual Motivation

In an environment where good software writing skills are scarce, it may seem as a puzzle that many software engineers – many located at prestigious firms – would donate their time to giving software to the community. This behavior becomes less surprising if we take seriously the gift-giving culture of software development (Rheingold, 1993; Raymond, 1998). This culture is expressed forcefully in the Open Source movement, and especially its GNU Public License (GPL).[3] This latter document, drafted by Richard Stallman (the founder of the Free

[2] See RedHat's website at http://www.redhat.com.
[3] See the GPL – the key provision is that any work that in whole or in part contains or is derived from the Program is

Software Foundation) and sympathetic legal advisers, provides the legal foundations that protect open-source software from being privatized by individuals or companies.

For many firms, such as Netscape (or its owner AOL), these provisions are too radical and prevent GPL protected software to be combined easily with private software modules, such as in server software systems. Thus, many open-source software packages provide free use, but subject to prohibitions on the use of specific modules and the incorporation of the software in proprietary products.

The motivations for the individual engineers are partly driven by norms of reciprocity, partly by reputation enhancement, partly by love of their work. The importance of reciprocity in the exchanges among members of the open source community has been recently documented by Lakhani and von Hippel (2000). The study of the support groups for the Apache web server shows that the most important reason for people posting answers on Usenet groups is the desire to help because they have been helped by others, or because they expect to need others' expertise in the future.

The diffusion of one's code to millions of users is akin to the professional accreditation that is found to be so important for the use of publications in the science-business networks of the pharmaceutical companies (Stern, 1999) or the role of patents in the electronics industry. The reward to an engineer is not just fame. As Ellen Ullman's (1997) deeply personal book about her work as a programmer shows, developers experience a strong personal satisfaction from creating "something that works." For a contribution to be seen as a sign of merit, the open-source project must be interesting to the engineering community.

---

to be licensed as a whole at no charge to all third parties. In other words, any software that contains a piece of 'free software' is to be distributed, in its entirety, as open source software.

Being an open source contributor also improves the vita of the software engineer, a valuable asset in today's labor market of high turnover. Individual motivations also include the value to participants to increase their market value (Lerner and Tirole, 2000). The open source movement has experimented with some new forms of remuneration. Increasingly, vendors and engineers are asking to move from a fee payment to royalties that share in the profit of the final product. Potentially, code can even be auctioned through public posting of the patch. The development of new payment policies will allow a far greater expansion of the willingness of engineers to participate in radical innovations. Similarly to the music industry, the incentive to solve remuneration issues is strong. New ways of tagging code to permit the identity of the source are already available. Indeed, Lessig (1999) argued that the technology to tag Internet content is greater than for other media, thus posing a problem for a legal regime that is too much prone toward intellectual property protection. It is already possible to sell software by auction, which can be tested and inspected, while preserving copyright protection.

Organizational Challenges

In addition to the incentives for individuals, the second explanation for the emergence of community-developed software points to the great gains to firms that are possible through the harnessing of the web for innovations. The modularization of software permits development through a series of complex adaptations (Baldwin and Clark, 1997). It is important to recognize that the possibility of exploiting distributed intelligence through complex adaptation requires a number of pre-existing conditions (Kogut and Metiu, 2001). A primary condition is the diffusion of Unix that was understood by a large number of engineers. Once this language had diffused, it was possible to assemble a more complicated product through the creative adaptation and

recombination of the existing knowledge base. The speed of development was largely the outcome of this combinative capability inherent in this community. Though Linux is not Unix but an operating system written in Unix, the inventory of stored knowledge in the form of Unix-encoded modules could be quickly adapted to support the Linux kernel.

However, community-based development is not entirely a complex system consisting of independent and equal individuals. First, Torvalds wrote an integral kernel. This decision in fact caused controversy in the development community, where there had been a preference for modularity even in the core operating code (See the appendix A to DiBona, Ockman and Stone, 1999). Second, there emerged a hierarchy by which a few prominent programmers took on responsibility for overseeing the development of the software for key functions. The degree of a priori hierarchical structuring in the allocation of software tasks was reflected in the hierarchy by which Torvalds would oversee every piece of software that went into the kernel. It was precisely this hierarchy that imposed the need for a more distributed governance, once it was discovered that "Linus doesn't scale," i.e., that he could become overwhelmed with the amount of new software submitted by developers (Young and Goldman Rohm, 1999).

The business models behind open-source firms are built on this very need for scalability. Firms are needed to provide the recourse to liable sources to ensure quality and warranty. A multi-billion dollar company, as well as the individual user, wants to know that there is a legal entity whose survival depends on the quality of the product, and who offers competent service. Firms who support and sell the community-developed software (a prominent example of which is Red Hat) are absolutely vital for the success of the product, because the consumer requires recourse for reliable and liable control for quality and service. Concerns over bad quality also miss the real impact of this new model. It is the variability in quality that has created the

possibility for companies to offer critical value-added services. As Chuck Murcko of the Apache Foundation pertinently observes, while there is a lot of bad code written in the open-source community, we don't know if this any more than the bad code contained in a proprietary software program.[4]

Linux is an important existence proof of the innovation-driven model. Distributed innovation, with no firms and no contracts, created a highly successful product. However, firms are still needed because the market for credibility still demands that companies be built to ensure that there is a contractual guarantee of performance. Open development does not eradicate the market, it builds it.

Implications

Linux is not the only instance of the innovation-driven model. Similar processes and organization can be found in the development of the Emacs Lisp library and the Apache server software. As Eric Raymond (1998) writes, innovation through the bazaar of scattered developers has proven itself capable to compete with development by the established firms' cathedral model of organization.

The information technologies facilitate collective endeavors at innovation, and they open the innovative process to more than company employees. Once it becomes clear that the lore of the innovator can be shifted to the community, the reliance on the global project model for the reduction of development costs is relaxed in favor of an open development model that exploits innovation on a global basis.

[4] Comments made at the workshop on global software development, Reginald H. Jones Center, Wharton School, December 1999.

Consider a firm in our study that makes office equipment. Historically, this equipment consisted of mechanical parts moving at high speeds, governed by electronics and increasingly microprocessors. The digital revolution converted the paper that the machines manipulated into electronic documents. To govern this process, the company develops proprietary software that requires massive allocation of resources. The open development model says that this firm may be able to do this not only faster, but also better. At a minimum, it can post information on its intranet to permit developers from all its multinational research and development sites to build interfaces with the operating system. The costs of communication and coordination are low, and a shared context is dependent upon the developer. The company incurs the fixed costs of supporting engineers who are permitted to dedicate some of their time to responding to web-posted inquiries.

The elements to this strategy are within the grasp of many companies. It requires a combination of the famed 3-M strategy to free up a substantial part of a developer's time to allocate to projects of his or her choice, plus the infrastructure support to permit an intranet dissemination of information. The strategy also requires tough rules about property rights. But remuneration for the innovating developer is not more difficult than the standard evaluation of the contribution of an individual to a team, though here the contribution is to the company community.

Even a large company like Microsoft has been able to adopt development characteristics generally ascribed to open source models. Cusumano and Selby (1995) explain that in order to encourage exchange of ideas, Microsoft builds software teams and cultivates developer networks within the company. In this sense, Microsoft seeks to capitalize on the benefits of open development by creating an internal community to appraise and debug the innovations of

software teams. Yourdon (1996) praises the company's practice of instituting the "push back method" whereby people challenge each other's ideas. An important consequence of this idea exchange is that many Microsoft project teams achieve significant commitment to the project's critical success factors. Yet, this model does not clearly address the many problems of working in a spatially distributed environment that still allows for spontaneous incremental improvement.

A more radical proposal is to post the operating system publicly. The cost of this posting is the concern over the loss of proprietary ownership. Some companies are already accepting this possibility for some of their projects. The most prominent example is IBM, who recently became the first Fortune 100 companies to achieve the status of full member in the open-source community. In December 1998, the company released Jikes' source code. Jikes is a Java source-to-bytecode compiler that is extremely fast. Within eight hours of the release, IBM received a solution to an error (i.e., a "patch") from a developer who is now a core team member on the project. IBM has demonstrated its strong commitment to the open-source community by devoting important resources to open-source projects. The firm pays salaries to open source developers, it devotes material resources to ensure the infrastructure of the projects, and installs open-source software on its products. Moreover, IBM has dedicated developers who work outside of IBM with the community at large, and its license was approved by the Open Source Initiative. The projects are hosted on a Linux server outside the IBM firewall. Dave Shields, who spearheaded the Jikes project within IBM, says that perhaps the most important aspect of IBM's approach is that the company realized that its initially proposed "steward" role is not needed. Currently five of the eight core team members are not IBMers, and authority is equally distributed among them.

One of the important challenges for companies regard the protection of property rights. However, it is more important issue to understand how firms will benefit from an open development strategy. The explicit strategy of Linux firms is a model that shifts revenues from selling the software (but not entirely; many of us will still prefer to have a warranty-registered CD just in case) to the provision of services, online support, and training. And the ability to mix open source with some proprietary software, such as Netscape has done with its Mozilla licensing policy, can still maintain an edge for competition in software products itself.

Conclusions

Open development is possible not just for software, but for all fields in which cooperation can be arranged by modules and there exists a wide understanding of a common language and culture. An obstacle to exploiting the open development model is the belief that innovation is the product of the individual genius located in economically advanced countries. But innovation has always been a mix of the individual and the group. And the emphasis on one or the other is a cultural interpretation. Both are usually present.

The open development model opens up the ability to contribute to innovation on a global basis. It recognizes that the distribution of natural intelligence does not correspond to the monopolization of innovation by the richest firms or richest countries. It is this gap between the distribution of ability and the distribution of opportunity that the web will force companies to recognize, and to realign their development strategies. For the young engineer in India, China, or Israel - who cannot or does not want to come to the Silicon Valley, or the Research Triangle, or Munich - is increasingly able to contribute to world innovation.

The consequences of the distributed innovation model for the development of offshore sites and remote regions are apparent. Not long ago, regions such as Bangalore and Hyderabad were mere providers of low-cost engineers ("body-shoppers") to clients from the rich countries. However, these regions have gradually gained the right to work on more challenging, innovative projects. The share of services performed at the client sites of total Indian software exports has decreased from 95% 1991-92, to 58% in 1999-2000 (Nasscom, 2000). Furthermore, in 2000, 185 of Fortune 500 companies outsourced their software requirements to India (Nasscom).

The new model of organizing has also important consequences for the future of the multinational corporation? An argument for what multinational firms do is that they act as efficient mechanisms for the creation and transfer of knowledge among countries (Kogut and Zander, 1993). This role is potentially substituted by the increased possibility to find and coordinate distributed intelligence through virtual communities. In these on-line communities, people find each other—either in the form of buyer and supplier, or in the form of participants to a virtual project.

However, it is easy to see that the multinational corporation will be needed even for projects where it does not have a competitive advantage in the coordination of work across borders. The principal characteristic of international markets remains the absence of a global regulatory and judicial order. This absence clearly influences such issues as the transmission of private data and rules regarding privacy (Kobrin, 2000). In cases where the protection of intellectual property is problematic and institutional protection against the state is an issue, the multinational corporation provides a private-ordering to assure the appropriation to contribution. In this view, the multinational corporation is not so much a governance mechanism destined to

resolve internal incentive conflicts, but a countervailing force in weak institutional regimes (Henisz, 2000).

References

Adelson, B., and E. Soloway. 1985. The Role of Domain Experience in Software Design. *IEEE Transactions on Software Engineering*, SE 11:1351-1360.

Arora, Ashish and Jai Asundi. 1999. Quality certification and the economics of contract software development: A study of the Indian software industry. Working Paper, NBER.

Baldwin, Carliss, and Kim Clark. 2000. *Design Rules. Vol. 1: The Power of Modularity*. Cambridge, MA: MIT Press.

Barnow, Burt S., John Trutko, and Robert Lerman. 1998. Skill Mismatches and Worker Shortages: The Problem and Appropriate Response. Draft Final Report, The Urban Institute.

Brooks, Frederick P. Jr. 1975. *The Mythical Man-Month*. Reading, MA: Addison-Wesley.

Buckley, Peter J. and Mark Casson. 1976. *The Future of the Multinational Enterprise*. New York: Holmes & Meier Publishers.

Cantwell, John. 1989. *Technological Innovation and Multinational Corporations*. Cambridge, MA: Blackwell.

Cusumano, Michael A. 1991. *Japan's Software Factories*. New York: Oxford University Press.

Cusumano, Michael A. and Richard W. Selby. 1995. *Microsoft Secrets*. New York: Free Press/Simon & Schuster.

Daft, Richard L. and Robert H. Lengel. 1986. Organizational Information Requirements, Media Richness and Structural Design. *Management Science*, 32:554-571.

Dempsey, Bert J., Debra Weiss, Paul Jones, and Jane Greenberg. 1999. A Quantitative Profile of a Community of Open Source Linux Developers. Report, UNC Open Source Research Team, School of Information and Library Science, University of North Carolina at Chapel Hill.

DiBona, Chris, Sam Ockman and Mark Stone. 1999. *Open Sources*. Sebastopol, CA: O'Reilly.

Ekman, Paul. 1985. *Telling Lies: Clues to Deceit in the Marketplace, Politics, and Marriage*. New York: W. W. Norton.

Glaser, Barney G. and Anselm L. Strauss. 1967. *The Discovery of Grounded Theory*. Aldine Publishing Company: New York.

Goffman, Erving. 1961. *Encounters. Two Studies in the Sociology of Interaction*. Indianapolis: The Bobbs-Merril Company.

Henisz, Witold J. 2000. The Institutional Environment for Multinational Investment. *Journal of Law, Economics, & Organization*, 16(2): 334-364.

Hennart, Jean-Francois. 1982. *A theory of the multinational enterprise*. Ann Arbor, Mich.: University of Michigan.

Information Technology Association of America. 1997. Help Wanted: The IT Workforce Gap at the Dawn of the New Century. Arlington, VA.

Kobrin, Stephen.

Kogut, Bruce and Udo Zander. 1993. Knowledge of the Firm and the Evolutionary Theory of the Multinational Enterprise. *Journal of International Business Studies*, 24: 625-645; reprinted in *Transforming International Organizations*, edited by William G. Egelhoff, Northampton, MA: Edward Elgar Publishing, 1997.

Kogut, Bruce and Anca Metiu. 2001. *Distributed Knowledge and Open Source Software*. Forthcoming, Condor.

Kraut, R. E., and L.A. Streeter. 1995. Coordination in Software Development. *Communications of the ACM*, 38: 69-81.

Krugman, Paul R. 1991. *Geography and Trade*. Leuven, Belgium: Leuven University Press and Cambridge, Mass.: MIT Press.

Lacity, M. C., L. P. Willcocks, and D. F. Feeny. 1996. The Value of Selective IT Sourcing. *Sloan Management Review*, Spring:13-25.

Lakhani, Karim and Eric von Hippel. 2000. How Open Source Software Works: 'Free' user-to-user assistance. Working paper #4117, MIT Sloan School of Management, Cambridge, MA.

Lave, Jean and Etienne Wenger. 1991. *Situated Learning*. Cambridge: Cambridge University Press.

Lerner, Josh and Jean Tirole. 2000. The Simple Economics of Open Source. Working Paper 7600, NBER.

Lessig, Laurence. 1999. *Code and Other Laws of Cyberspace*. New York: Basic Books.

McConnell, Steve. 1999. Brooks' Law Repealed. *IEEE Software* Nov-Dec.: 6-8.

McGrath, Joseph E. and Andrea B. Hollingshead. 1994. *Groups Interacting with Technology*. Thousand Oaks: Sage Publications.

McKenney, James L., Michael H. Zack, and Victor S. Doherty. 1992. Complementary Communication Media: A Comparison of Electronic Mail and Face-To-Face Communication in

a Programming Team. In Nitin Nohria and Robert G. Eccles, eds., *Networks and Organizations*. Boston, MA: Harvard Business School Press.

McManus, J. 1972. The theory of the international firm. In G. Paquet, editor, *The multinational firm and the nation state*. Ontario, Canada: Collier Macmillan Canada.

Metiu, Anca. 2000. Owning the Code: Creativity in Global Software Development. Working Paper, the Wharton School.

Miles, Matthew B. and A. Michael Huberman. 1984. *Qualitative Data Analysis*. Beverly Hills, CA: Sage.

Nasscom and McKinsey Study Report of the Indian IT Industry. 1999. See http://www.nasscom.org/

Nelson, P., W. Richmond, and A. Seidmann. 1996. Two Dimensions of Software Acquisition. *Communications of the ACM*, 39:29-35.

Nohria, Nitin and Robert G. Eccles. 1992. Face-to-Face: Making Network Organizations Work. In Nitin Nohria and Robert G. Eccles, eds., *Networks and Organizations*. Boston, MA: Harvard Business School Press.

Olson, Margrethe H. and Sara A. Bly. 1991. The Portland Experience: A Report on a Distributed Research Group. *International Journal of Man-Machine Studies*, 34:211-228.

O'Riain, Sean. 1999. Net-Working for a Living: Irish Software Developers in the Global Workplace. Forthcoming in M.Burawoy et al., eds., *Global Ethnography*. Berkeley: University of California Press.

Orlikowski, Wanda J. and Joanne Yates. 1994. Genre Repertoire: Examining the Structuring of Communicative Practices in Organizations. *Administrative Science Quarterly*, 39(4): 541-574.

Perlow, Leslie. 1997. *Finding Time: How Corporations, Individuals and Families Can Benefit form New Work Practices*. Ithaca, NY: Cornell University Press.
Perlow, Leslie. 1999. The Time Famine: Toward a Sociology of Work Time. *Administrative Science Quarterly*, 44: 57-81.

Perrow, Charles. 1970. *Organizational Analysis: A Sociological View*. Belmont, CA: Wadsworth.

Raymond, Eric S. 1999. The Cathedral and the Bazzar. See http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html#toc16

Rheingold, Howard. 1993. *The Virtual Community: Homesteading the Electronic Frontier*. Reading, MA: Addison-Wesley.

Tang, John C. 1991. Findings from Observational Studies of Collaborative Work. *International Journal of Man-Machine Studies*, 34:143-160.

Stern, Scott 1999. Do Scientists Pay to Be Scientists. Working Paper, NBER.

Ullman, Ellen. 1007. *Close to the Machine: Technophilia and Its Discontents*. New York: City Lights.

Von Hippel, Eric. 1994. 'Sticky information' and the Locus of Problem Solving: Implications for Innovation. *Management Science*, 40: 429-439.

Walz, D. B., J. J. Elam, and B. Curtis. 1993. Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. *Communications of the ACM*, 36: 63-76.

Wang, Eric T. G., Terry Barron, and Abraham Seidmann. 1997. Contracting Structures for Custom Software Development: The Impacts of Informational Rents and Uncertainty on Internal Development and Outsourcing. *Management Science*, 43:1726-1744.

United Nations, *World Investment Report*, 2000. New York: United Nations.

US Census Bureau. 1999. Services Annual Survey: Information Sector Services.

Yoffie, David B. and Michael A. Cusumano. 1999. Judo Strategy: The Competitive Dynamics of Internet Time. *Harvard Business Review*, vol. 77 pp. 70-81.

Young, Robert, and Wendy Goldman Rohm. 1999. *Under The Radar: How RedHat Changed the Software Business – and Took Microsoft By Surprise*. Scottsdale, AZ: The Coriolis Group.

Yourdon, Edward. 1996. *Rise and Resurrection of the American Programmer*. Upper Saddle River, NJ: Prentice Hall.

Zaheer, Srilata. 1999. Time-zone Economies and Managerial Work in a Global World. Forthcoming in P.C. Earley and H. Singh, eds., *Innovations in International Management*. Thousand Oaks: Sage Publications.