UC Santa Cruz
Working Papers

Title
Information Technology: History, Practice and Implications for Development

Permalink
https://escholarship.org/uc/item/951709tx

Author
Eischen, Kyle

Publication Date
2000-11-01

eScholarship.org Powered by the California Digital Library
University of California

CENTER FOR GLOBAL, INTERNATIONAL
& REGIONAL STUDIES
University of California, Santa Cruz • 156 Merrill College • Santa Cruz, CA 95064 http://www2.ucsc.edu/cgirs
831.459.2833 (Voice) • 831.459.3125 (Fax) global@cats.ucsc.edu

Information Technology: History, Practice and Implications for Development

Kyle Eischen
keischen@cats.ucsc.edu

Center for Global, International and Regional Studies
and
The Department of Sociology
University of California, Santa Cruz

November 2000

CGIRS Working Paper Series – WP#2000-4
Center for Global International & Regional Studies
http://www2.ucsc.edu/cgirs
University of California, Santa Cruz

They are called computers, simply because computation is the only significant job that has so far been given to them. The name has somewhat obscured the fact that they are capable of much greater generality…To describe its potentialities, the computer needs a new name. Perhaps as good a name as any is 'information machine.'

Louis Ridenour, 1952

Science, as well as technology, will in the near and farther future increasingly turn from problems of intensity, substance, and energy, to problems of structure, organization, information, and control

John Von Neumann, 1949

Software translates languages on Web

Lotus Ireland, a subsidiary of International Business Machines Corp., said Wednesday it had developed a new software product allowing people speaking different languages to communicate across the Web. Lotus Translation Services for Sametime translates Web chat and communications as typed in real time, and will go on sale in September. The software employs Java technologies.

San Jose Mercury TechTicker, June/22/00

I OVERVIEW 4

II THEORIES OF INFORMATION TECHNOLOGY: UNDERSTANDING IT AS PROCESS, PRODUCT AND INDUSTRY 5

III IT HISTORY 8

i Algorithmic Logics and IT 8
ii The Development of Machine Computers 9
iii Modern Computing 12
iv The Development of the Computer Industry 13
v From Vacuum Tube to Microprocessor 14
vi The Development of the "Information Machine" 15
vii Information Technology Fully Formed: The "Internet" 18
viii Post-PC Era 22
ix Software is IT: The Bricklayers of the Networked Society 22

IV SOFTWARE AS PROCESS, PRODUCT AND INDUSTRY: DOMAIN KNOWLEDGE, EMBEDDED INFORMATION AND SOFTWAREISM 24

i Defining Software Development 27
a Software Development Processes 27
b Software Rationalization Considered 30
ii The Impact of Software and Software Practice in an Information Society 33
a The Size of the Global Market 34
b Software Dependent Systems 35
c Unique Software Based Practices 36
iii Softwareism (!?): A Working Hypothesis on the Impact of Software in the Global Environment 37

V IT AND DEVELOPMENT: CONSIDERING STATE-LED REGIONAL HARDWARE AND SOFTWARE STRATEGIES 42

i Features of Innovative Regions: Developing a Matrix of Comparison 45
ii High-Technology Hardware Focused Development Strategies 47
iii High-Technology Software Focused Development Strategies 51
iv Building a Software Region: Opportunities and Limitations 58
a Potential Consequences of Software-Focused Development 59

VI IN CONCLUSION: DEVELOPING MODELS, METHODS AND RESEARCH PROGRAMS TO UNDERSTAND IT AND DEVELOPMENT 60

VII BIBLIOGRAPHY 64

I Overview

The presence of information technology (IT) within the global environment has been extensively studied, but its structure and impact has not been rigorously explained. In many ways, this failure to explain IT stems from a basic lack of understanding the technology itself, both in terms of its historic development and its basic production patterns. Such an explanation is important exactly because its absence limits an analysis of global processes generally, and the discussion of “information societies and economies” in particular. If information technologies are central in structuring new global trends, then it is an empirical necessity to detail the social and economic impacts derived from such technologies. Such impacts can not be understood, or at best only partially, if the basic conceptualizations and processes underlying IT are not defined.

The purpose of this outline is to step back from more macro approaches and consider information technology from first principles. The aim is to clarify and define both the informational and technological aspects of IT, then link these into considerations of the potential impact on economic and social development. The discussion is framed in four broad sections. First, current theorizations of information technology are considered. This details the current weakness of analysis to outline if and how IT is different from other forms of industrialization. The failure of such analysis stems from not understanding IT as simultaneously a process, product and industry. Approaching IT from each of these viewpoints helps clarify why “high-technology” and information technology sectors are distinct, how informational and industrial patterns differ, and how such differences play out on regional and local levels.

This is followed in the second section by a consideration of the historic development of IT as a technology and industry from conceptualization to implementation. Such an analysis offers key insights into how the conceptualization of an “information machine”, and the basic mathematical tools to support such a development, preceded the actual physical development of computing machines by at least a century. It also shows how the fulfillment of this conceptualization in reality is what structures the current age as informational, rather than computational. Looking at IT historically, it possible to trace the general informational concepts that frame IT, especially in terms of algorithmic structures, as well as understand why both the technology and industry have come to be dominated by software as the central sector and source of innovation within the industry.

The third section seeks to expand on the increasing significance of software in IT as process, product and industry. In many ways, understanding these aspects of software highlights the basic informational aspects of IT. For computers to function, they need both information and tools to process that information. The conceptual tools for this, in the form of binary structures and algorithmic logic, have existed from the beginning of computing. Software currently is the primary form for applying these conceptual tools to social and economic reality. Software is the means in which both knowledge and the tools that process that knowledge are structured, transformed and transmitted. This implies that software processes are a potentially central form, both within IT and society overall, of transforming and structuring social knowledge into commercial products and authority. This opens up a consideration of software, and software processes, as the ideal type of production process in an information society.

The final section considers the nature of innovative IT regions through a comparison of hardware and software focused development strategies. As software becomes more important in the global economy, regions will attempt to transition from previous hardware strategies to new state-led software focused initiatives. Given the distinct nature of software processes and industry structures, how new regions establish and sustain IT development will be a crucial question of coming decades. A simplified matrix of innovative regions offers a heuristic tool to evaluate what resources and capacities either IT sector requires and generates. This opens up the possibility of separating the general global trends that shape all regional IT developments from the distinct regional resources and capacities that promote sustained innovative environments. In this way, the comparison helps clarify the opportunities and limitations of software focused initiatives as quite distinct from hardware focused strategies.

II Theories of Information Technology: Understanding IT as Process, Product and Industry

Information technology (IT) is an assumed, but frequently poorly or unexplained force within analysis of the global environment. Much of this failure stems from a lack of clarification and definition of what exactly is meant by IT. On a general level, analysis often fails to clarify what are the informational and technological aspects of IT, and what overriding trends or structures link them together. In other words, there is a failure to establish what the basic processes and technologies are that define IT as a distinct and defined force in the global environment. This is visible in the confusion in terminology between "high technology" and information technology. Much of the literature seems to assume that these are equivalent terms, rather than pausing to consider that they actual might define separate processes, products or industries. This leads to a more specific failure to distinguish between sectors within IT, such as hardware and software. If they are both IT industries, similar policies, industry patterns, firm organizational structures, and production processes should characterize them both. However, without a clear definition of what IT is in general, it becomes extremely difficult to determine what connects and separates various IT sectors.

Yet, if information technologies are a central factor within the global environment, such definitions are essential to elaborating on various divergent global social and economic trends. As Rosenberg states, "this is because the specific characteristics of certain technologies have ramifications for economic phenomena that cannot be understood without a close examination of these characteristics (p. vii)." However, it is exactly these characteristics that are missing in much of the analysis of information technologies. Moving inside the "black box" of IT, it is possible to consider the "micro foundations of economic dynamics (Dosi 1984: 1)" that link technological processes with the macroeconomic and social forces that define a globalized environment. Such an approach offers a more robust and detailed analysis of well considered aspects of the global economy including innovation, networks, the role of the state and new social inequalities, without either slipping into technological determinism or dismissing the social implications of IT all together.

Current theorizations, failing to begin from this micro-foundational level, tend to analyze IT broadly and from either industry, product or process perspectives. General theorizations of globalization recognize the importance of technology, but tend to approach it through a consideration of specific industry globalization patterns (Held et al 1999, Dicken 1998). These tend to focus on industries like finance (Sassen 1996), integrated circuits (Dicken 1998, Henderson 1989), software (Mowery 1996) or electronics (Ernst 1997). This analysis tends to use the globalization of IT industries as a proxy for the impact of information technology in the global environment. Such analysis suffers from not distinguishing between the internationalization of industries (which is not a new phenomenon) and the distinct characteristics of IT products and processes that do structure global trends in a historically unique manner. The analysis captures well the quantitative aspects of global industries (trade flows, FDI, employment, firm location), but does not capture the qualitative aspects that really distinguish IT patterns. In other words, this analysis makes no connection to the unique IT processes and products that influence the macro trends they analyze.

Another set of analysis tends to view IT as a series of products (or goods and services) that impact upon social and economic relationships. Much of this analysis revolves around the impact of specific information technologies, especially the Internet, on economies and societies (Cairncross 1997, Kelly 1998, Taylor et al 1997, Coyle 1998). A sub-category of such analysis can also be found in aspects of the business press focusing on "new economy" trends and impacts (i.e. Red Herring, Wired, Business 2.0, FastCompany), as well as in considerations of the digital divide (Digital Economy 2000). Such analysis assumes that IT has an impact by simply being produced, without either an empirical understanding of the relative weight of IT in society nor an understanding of the processes that would shape such an impact. This is not to imply that such analysis is wrong. Much of it captures quite well the technological aspects of key information technologies, and elaborates on the potential impacts of such products. However, the lack of quantifiable data linked to broader theories make it difficult to evaluate how IT products are shaped by specific institutions (firms and states) and social forces (education, income, norms, culture), as well as how widespread their impacts are or may be.

The final entry point to defining IT comes from process perspectives. Simply, the process analysis considers the informational aspect of IT, and focuses on defining how societies and economies are transformed by the application of information technologies. Within this, there are two broad categories: macro analysis of how IT establishes a new conception of social and economic reality, and more detailed meso-level considerations of how development processes are structured geographically and organizational. In the conceptual case, there is an extensive literature that seeks to understand the shift that is signaled by a move from a physical to an informational-based environment. Such analysis focuses on the nature of a "digital society" (Negroponte 1995), the role of knowledge as the central resource (Drucker 1993), and the spatial and symbolic impacts of such developments (Lash and Urry 1994). This literature is complemented by the second category that analysis the institutional and geographic structures that transmit and transform information on regional (Saxenian 1999) and global (Gordon 1994b, Borrus 1993) levels through new organizational forms (Castells 1995, Amin 1994). However, while analytically rich, each category fails to link the conceptual and meso-level changes to changes in concrete behavior and practice. The literature analyzes how information technologies impact upon the global environment generally, but does not offer robust analysis or predictions of how firms, governments and individuals structure micro-level responses to these general trends. This failure to effectively link to more grounded analysis weakens the ability to explain the contradictory trends that characterize IT processes in the global environment.

Overall, each of these entry points into the analysis of IT fails to be informed by the others. A more accurate understanding of IT in the global environment requires a combination of industry, product and process approaches. Broader industry trends would be greatly enriched by an understanding of the technical characteristics of the products and services that structure the strategies and capacities of specific industries. How such industries are structured on a firm, regional and global level clearly is linked to the production processes that transform and commercialize such information. Historical analysis of IT products and industries would open up understanding of longer term trends that structure conceptions of IT, as well as help define specifically what IT is. Importantly, a combination of the three literatures would help link information and technology to global and local practices. This in turn would help clarify and define the impact, structure and location of IT sectors in the global environment.

In other words, approaching IT from process, product and industry viewpoints creates a much richer methodology for understanding the impact of IT on economic and social processes. Arguably, it is also the only way to capture both the technical and social aspects of IT simultaneously (Agre and Schuler 1997, Agre 1995b). It also offers a means for comparing the characteristics of IT and "information societies" with other industrial transformations to determine what exactly is "new" in the new economy (Cohen et al 2000). The next section begins a first step of such an analysis through a history of the conceptual, technical and industry trends of IT.

III IT History

i Algorithmic Logics and IT

The simplest definition of IT is one word: algorithm. Simply defined, an algorithm is a procedural description of how something should be accomplished. Bureaucracies are socially institutionalized forms of this, as are the patterns that govern basic biological processes (Ridley 1999). What has developed over that last three hundred years is the creation of tools to describe and replicate both these naturally occurring and constructed patterns mathematically. The most widely applied example of this is within the IT field today, both in hardware and software processes.

Algorithms arose from the study of logic tracing back to Aristotle. The word itself is derived from the surname of Ja'far Mohammad Ben Musa al-Khowarazmi, the ninth century Muslim mathematician who wrote "The Science of Restoration and Reduction" that transformed algebra and brought Arabic numerals to Europe (Leonard 2000). Algorithmic logic was revived as a defined and respectable field of knowledge by Leibniz, the co-creator of the calculus, at the end of the 17th century. Leibniz offered a very simply but powerful idea: the entire universe can be described as comprised of god and nothingness (Berlinski 2000). In other words, all of life follows a discrete binary system that can be modeled, or coded, within a logical algebraic framework. As such, real-world processes could be mapped using mathematical symbols, if the underlying algorithms could be identified. This opened the theoretical possibility of modeling both the social processes of bureaucracies and the basic sequence of DNA, among others, as mathematical abstractions.

Leibniz insight into algorithmic logic and binary systems are at the heart of IT as we have come to conceive it (Berlinski 2000). IT is essentially defining a logical algebraic function that produces consistent outcomes for specific processes then codifying them in either software or hardware formats (Mitchell 1998). However, the actual application of this conceptualization into practical working systems took almost three hundred years. The following history is a rough outline of how this occurred through a mixed technical and industry viewpoint, both of which have influenced the form that Leibniz' conception has taken within information technology. The history begins first with the desire for a "machine computer", then the application of computing to information processing under business machine manufacturers, the actual building of binary coded mainframe machines the 1960's, and finally the extension of such systems through the rise of personal computing and internets.

ii The Development of Machine Computers

The actual attempts to apply algorithmic logic to automate problem solving began in the industrial revolution. The industrial logic of the time led to numerous attempts to create "smart machines". As complex and precise mathematical calculations increasingly came to be essential to a modern industrial economy, machine calculation came to be seen as a rational solution. Information, like any other industrial activity, needed to be processed in ever-greater quantities and efficiencies. At the time, the mathematical computations for such key publications as the Nautical Almanac, showing tide charts, were done by teams of human "computers". The quantity and diversity of information needed to be processed, along with the high incidence of human error, created a general push for the creation of mechanical computers (Campbell-Kelly and Aspry 1996).

The preeminent example of this is Charles Babbage's attempt to develop the Difference Engine in England in the 1830's in order to mechanically produce solutions for tide-tables accurately and efficiently. The conception, like the underlying mathematics, was clearly understood, with a defined sequence of mathematical steps (an algorithm) that would produce the needed tables accurately and consistently. However, the actual construction of the "Engine" (the name itself demonstrating the logic of the era) proved almost beyond the technical capabilities of the time. A small working model was produced, but it was unable to fulfill the table-making functions for which it had been funded to solve. Importantly, Babbage also conceived, though never completed, the concept of an "Analytical Engine". The Difference Engine was inherently limited by its dedicated mechanical design to solving one specific mathematical problem. The Analytical Engine was conceived as a programmable machine that could perform multiple mathematical functions rather than merely pre-defined calculations (Moschovitis et al 1999). Conceptually, this was the first attempt to build what would now be recognized as a computer. It would be another hundred years before a fully functioning programmable calculator as envisioned by Babbage would be built.

The increasing demand for information processing came to dominate the development of mechanical computing, overriding Babbage's general idea of a multi-purpose calculating machine. Business equipment applications and manufacturers came to dominate the search for mechanical means to process and tabulate information. Herman Hollerith developed a mechanical system for processing census data that was implemented in the US in 1890. The system tabulated the census data in six weeks, as compared to seven years for the previous census[1]. Hollerith went on to form the Tabulating Machine Company which became the foundations of IBM. The computing industry would continue to be dominated by business equipment firms developing machines to handle specific information processing needs, as opposed to generating or manipulating general information, until the advent of the Second World War (Campbell-Kelly and Aspry 1996).

[1] The basic machines built by Hollerith still work 110 years later. A working edition is on display at the Computer History Museum at Moffett Field (www. computerhistory.org). It is also interesting to note that other tabulating machines from this time, particular voting machines, are still in use to this day, even as the companies that made them have gone out of business (Wall Street Journal, November 9th, 2000).

The search into a mechanical means to manipulate and generate information did continue simultaneously, but was limited to extremely complex problems involving a few key users concerned with large industrial projects or government-related programs. Still facing the limits of human computers in a complex industrial world, mechanical computing increasingly focused on analog machines. Processes that were too complex to mathematically define, or be handled by existing mechanical computation, could be modeled through mechanical analogues of the process itself. These were machines that could create mechanical analogies of complex systems, such as dams, electrical networks or tides, that could then be replicated and scaled efficiently (Campbell-Kelly and Aspry 1996). Though analog computing was inherently limited by the need to construct a new model for each process, it did serve the specific needs of large-scale users. Such computing reached its peak impact between the two World Wars, with increasingly large and dedicated models helping analyze increasingly complex engineering and scientific problems.

A key development in analog computing occurred in 1931. Vannevar Bush of MIT was able to develop a “differential analyzer” that could perform a whole series of engineering and science problems based on differential equations. While diverse in application, the machine was not a real computer in the sense of performing and generating calculations. It was still an analog machine, modeling natural processes rather than manipulating numbers themselves, even if it could model multiple analog machines (Edwards 1996). As such the application of the machine outside of predefined engineering or science problems or to problems that could not be physically modeled (astronomy, physics, weather, code breaking) was impossible[2].

Only at the very end of WWII would the binary logic of Leibniz, Babbage’s general purpose computing machine concept, Bush’s practical model of a multi-purpose machine, the experience of business machine firms in robust information processing equipment, and the need to generate pure mathematical information come together to form the modern computer architecture. Even then, it would be another two decades before the use of computers would be integrated with the business equipment industry to find a widespread base for the use of computing power, and another four decades before computers were spread to the population as a whole.

[2] One of the little now stories of World War II, hidden due to national security concerns, was the continued and important role of human computers, in particular women, in the code breaking and other high-level mathematical computing needs. See Campbell-Kelly and William Aspray 1996, Edwards 1996.

iii Modern Computing

In August of 1944, Hathaway Aiken and a team at Harvard completed the Mark I, the first fully programmable computer to come into being. The five-ton machine worked through the inputting of operational codes on lengths of paper tape, and was designed to produce ballistics computations and code breaking for the US Navy. The design by Aiken was implemented by IBM and essentially consisted of a row of electro-mechanical punch-card machines. Being both programmable and automatic in performing general mathematical calculations, it was in many ways the fulfillment of Babbage's original design for the Analytical Engine.

Importantly, the development of the Harvard Mark I represents a convergence of technologies, ideas and institutions that would dominate many of the basic features of information technology for the next thirty years. On the technical side, the Mark I represented the first use of digital architectures and data, though decimal and not binary. In other words, information was transformed into discrete mathematical symbols that became both an input and output of the computing process for the first time. It also marked the full engagement of IBM into the field of computing, creating a blending of computing and business machines for the first time. The practical skills developed by IBM engineers in designing information-processing equipment for industrial uses blended extremely well with the experimental and theoretical environment of leading research universities. This convergence was further stimulated by the interests and very substantial economic support of the US military that viewed computing as a means to an end in support of initially World War II and the then the Cold War (Edwards 1996).

The final technical architecture for the computer as currently defined was formulated at the Moore School of Electrical Engineering at the University of Pennsylvania in 1944 and 1945. Operating under the auspices of the National Defense Research Council (NDRC), a team led by John Mauchly and Wallace Eckert had been commissioned to develop a computer to calculate ballistic tables for the US Navy. Completed in 1944, ENIAC was the world's first electronic computer, comprised of 18,000 vacuum tubes, 10,000 capacitors, 6,000 switches, and 1,500 relays. Because it performed calculations at 5,000 operations per second, faster than programs could be paper fed, it had to be programmed via hardwiring that called for physically wiring the machine to determine the circuitry of the programming logic.

The Moore school's second version, EDVAC, combined advances in electronics with the programmable flexibility of the Mark I to finally overcome many of the basic challenges facing computing. Spurred on by John Von Neumann, a member of the Institute for Advanced Study at Princeton and a consultant to the Manhattan Project, the EDVAC sought to solve the problems of limited memory (limiting the ability to store programs), too many vacuum tubes (creating tremendous heat, instability and high-energy consumption), manual and wired programming (making running new programs time consuming and tedious), and the use of decimal numbers (limiting the amount of information that could be processed and stored). Though never completed, the EDVAC design outlined by Von Neuman, Eckert and Mauchly defined the base architecture that all computers have to this day: stored-programs, binary logic of programs and computation, basic input and output units, a control unit and an arithmetic unit (Edwards 1996, Campbell-Kelly and Aspry 1996)[3]. This structure provided the road map from which all computers would develop in the following decades. The government would continue to sponsor widespread research and development of computer technology, but the early EDVAC pioneers also began the first real efforts to commercialize digital, electronic, programmable computer technology after the end of the war.

iv The Development of the Computer Industry

Throughout the 1950's and 1960's the computer industry developed within the broad framework of implementing the vision of the EDVAC in reality and expanding the efficiency and robustness of the components themselves. Once again the commercial market came to be dominated by the demand for large scale mainframe data-processing machines, as related to solving business or information processing related problems. While the invention of the stored-program computer created a potential split of hardware and software, the development of the industry focused on complete hardware and software solutions developed for specific end-users. In other words, while the basic innovations of the EDVAC design created the potential for a multi-purpose computer as exists today, the industry standard of packaged hardware/software solutions created machines that were far more similar to their dedicated analog predecessors than to the flexible machines in existence currently. The vast majority of software was custom built for specific main frame systems, almost exclusively by the producers of the hardware themselves, focusing on specific information processing needs. The use of electronic

[3] Basic computer architecture is still referred to as Von Neuman architecture, though it is generally agreed that Eckert and Mauchly were integral to the design.

circuits gave amazing speed, but a flexible, multi-purpose "information machine" was still much more theory than reality.

During these decades, computing development was like a building a new cathedral each time new technology emerged or systems were modified as a user's needs changed (Raymond 1998). Complete systems of hardware and machine specific software would be developed to address each end users demands. Computing was for only large-scale established projects with deep budgets. End-users were locked into both hardware and software that were not transferable to other systems or uses even within the same family of computers. Naturally government, in particular the military, and large corporations were the central users of computing power (Edwards 1996, Guice 1997). Of the business-computing companies, IBM dominated the private market for combined solutions of hardware and software through the 1960's, with a roughly 70% market share. It is important to note that IBM, though dominant in the market, did not have a clear lead in computing technology over other mainframe companies. IBM provided trust and stability for firms buying its computer systems, with greater resources to bear on marketing and maintenance, but no real sustained advantage in the technology itself (Campbell-Kelly and Aspry 1996). Even more important, innovations in the industry tended to come from private, as opposed to government or university, supported research.

v From Vacuum Tube to Microprocessor

The most significant events outside of the development of computing as a business machine revolved around the transformation of the basic architecture and structure of computer processing. All computers from the ENIAC forward suffered from the size, heat and energy constraints imposed by vacuum tubes for computation, that is the creation of the basic circuits that could manipulate binary language. In 1946, William Shockley, John Bardeen and Walter Brattain of Bell Labs created the first transistor, a solid state semiconductor that acted as a reliable and efficient amplifying and switching circuit. By 1958, IBM was producing transistor based business machines. In 1959, Robert Noyce of Fairchild Semiconductor and Jack Kilby of Texas Instruments independently create the first integrated circuit (IC) (Moschovitis et al 1999, Braun and Macdonald 1982). The IC is a seminal moment in computing history, because it allows the integration of multiple transistors on a single piece of silicon. A single silicon chip was able to contain a room full of computing power on a single chip. In other words, the basic hardwired programming structures that had taken up a room in the early ENIAC could now be fit in a desktop calculator.

The ability to miniaturize computing opened the door to wide-scale application of computing power outside of the traditional government and private sector applications. Even more important, Moore's Law – named after Gordon Moore, one of the founders of Intel – states that computing power, that is the number of transistors able to be placed on a single semiconductor, follows a linear trend which doubles every two years (Moschovitis et al 1999). This means that both miniaturization and computing power have had constant and predictable development trajectories over the last thirty years, with projects previously unfeasible because of size or cost concerns becoming possible within a short time frame.

The ultimate development in this process was the creation of the microprocessor in 1971 by Intel. The microprocessor combined all the elements for computation on one single chip. Rather than have multiple chips each dedicated to a "hard coded" process, the microprocessor enabled one chip to serve multiple functions (Braun and Macdonald 1982). As but one example of the implications of this transformation in size and computing power, calculators moved from a low-volume high-end business product composed of dozens of chips (many customized) to a mass market educational and home product composed of one single IC (Campbell-Kelly and Aspry 1996). After 150 years, the development of the IC finally brought Babbage's basic conception of a multipurpose programmable machine computer to society for the first time.

vi The Development of the ÒInformation MachineÓ

The technology transformation provided by the IC pushed the creation of a multi-purpose information machine that could process, manipulate and most importantly create information in multiple formats. This trend would progress rapidly with the widespread development of personal computers and the increasing dominance of software as a distinct driver of technological change within the computing industry. Again, IBM played a key role in this development. Like many dominant firms throughout IT history, IBM's corporate strategic decisions have shaped the development of the computer industry even more than its technological advances. IBM was at the center of two such strategic decisions during the 1960's (Campbell-Kelly and Aspry 1996).

First, the introduction in 1964 of the IBM System/360was the first computer that enabled software programs written for one computer to be used with all other computers within the system family. For the first time, software was scalable as an organization's computer needs expanded, with software becoming transferable to higher levels of computing power. Changing hardware no longer entailed the creation of entirely new software products or the complete transformation of existing systems. This dramatically reduced the cost and maintenance of software development, and let software developers begin to focus on the expansion and reuse of existing software programs for the first time, even if still limited to a specific hardware architecture.

More importantly it opened the possibility of separating hardware and software purchase decisions, if hardware and software development ever became unbundled. This is exactly what happened, and signaled IBM's second industry changing strategic decision. In 1969/1970 IBM decided to unbundle its software and hardware sales, opening the market for independent software producers to compete to supply the nearly 70% of the market IBM controlled.[4] The combination of these two events led to the development of a software industry focused on providing previously bundled services and products, especially the initial development of a packaged software sector.

The simultaneous miniaturization of ICs and the formation of an independent software sector converged to fundamentally reorganize the computer industry in 1977 with the release of the Apple II, the world's first true personal computer. The basic form of the PC, as it came to be structured in the Apple II and the industry overall, was created at Xerox PARC in the early 1970's, then slowly brought to market throughout the next decade (Moschovitis et al 1999). The graphical user interface (GUI), the "mouse", wysiwig visualization (what you see is what you get), and the Ethernet were all developed within the Palo Alto lab, but later commercialized by other companies throughout Silicon Valley. The pattern established by the introduction of the PC became characteristic of the industry overall. Entire new industries would develop virtually overnight, spurred by innovation in private or university labs, that were commercialized by new firms that quickly challenged existing dominant players and business models (Campbell-Kelly and Aspry 1996).

[4] There are competing versions of why IBM chose to unbundle software and hardware. The two most predominate explanations focus on pending anti-trust action by the Federal government and the internal cost of developing full hardware and software systems. Given the cost of developing System 360, a project that almost bankrupt IBM, the second seems

The PC also signaled the emergence of software as a key industry driver (Hoch et al 2000, Steinmuller 1996). The market for software, and prepackaged solutions in particular, has slowly come to shape the development of the computer industry overall[5]. From the introduction of the personal computer in the mid-1970's forward, the market for packaged software has increasingly expanded. Even if hardware came with proprietary software operating systems, such as the Apple OS, IBM's OS/2 or MS-DOS, independent software vendors provided additional programs to add functionality to these systems. Increasingly through the early 1980's, these packaged software products became essential determinants of hardware acceptance in the market. Programs such as Microsoft Word, VisiCalc, Lotus 123, and PageMaker established whole new industries and led to the promotion of the specific hardware systems they ran on[6].

By the early 1990's, the commodification of computing power and memory beyond basic user needs led to software being determinate in the range of uses and possibility of computing overall. The trend has continued with recent surveys indicating that PC owners have on average over 137 software titles on their home PCs (WSJ, 10-Aug-00). The computer has truly become an "information machine", able to perform multiple functions of processing and generating information.

Additionally, while customized software solutions are still prevalent for large systems, the expansion of packaged software has slowly continued to expand throughout the industry. By the early 1990's, the expansion of computing in general, and the internet in particular, gave rise to packaged software solutions based in Enterprise Resource Planning, databases and internet communications for large scale, as opposed to individual, users for the first time. Many of these programs took the shape of packaged systems that were sold broadly but that could be customized to individual user needs. The full direction of the industry since IBM's initial decision to unbundle software and hardware is demonstrated by the transformation of business machine or computer makers into predominantly software companies. Firms like Oracle, Sun Microsystems and IBM are fully or in large part business

---

more plausible. IBM, and other companies, considered hardware to be the driving force of the computer industry. Arguably, this has been an accurate understanding of the computer industry overall until the mid-1990s.

[5] Ironically, in many ways Apple's business model was a replication of the hardware/software integration model that IBM had abandoned in the 1960's on a PC scale. And even Apple's success was based on its proprietary software operating system and unique software applications and not unique hardware offerings or manufacturing ability.

[6] The best example of this is the strong belief that both VisiCalc and PageMaker, and to a lesser degree Microsoft Word, were crucial to establishing the early dominance of Apple. In some markets, publishing for example, this early dominance has never been lost.

software companies (Hoch et al 2000:27). The recent rise of firms specializing in B2B Internet commerce software solutions are merely inheritors of this overall trend.

The essential aspect of the rise of an independent software industry was the movement away from fixed, preprogrammed solutions of either systems or ICs. The flexibility and versatility that software enabled became the key aspect of all computer based systems. As ICs have been embedded in an ever-wider range of products, this trend has continuously moved into the most miniature of products, with everything from wristwatches to cell phones containing basic software code in place of hardwired IC solutions. Software has increasingly come to dominate the direction of computer technology, and essentially signifies the transformation of the computer industry to what is now called information technology. The other significant computing technology trend from the 1960's, the internet, has only further expanded on this basic pattern of embedded computing power, given flexibility and dynamism by software, to create ever greater abilities to process and generate information in multiple forms.

vii Information Technology Fully Formed: The ÒInternetÓ

In the late 1960's, prior to the development of a versatile and affordable "information machine" in the form of the PC, computing systems were complex, dedicated, customized and expensive. Leading research institutes throughout the United States all had proprietary systems focused on specific computational problems. The inability to share information or to inexpensively recreate such systems pushed the Defense Advanced Research Project Agency (DARPA) to fund research into creating a system to link the computing centers and allow researchers to share information (Guice 1997). The project became to be known as the ARPAnet, a dedicated communications systems linking key research institutes throughout the US. Built by a combined team of DARPA officials and Bolt, Beranek and Newman (BBN) of Cambridge, Massachusetts, the original ARPAnet when online in December 1969 connecting UCLA, UCSB, SRI International and the University of Utah (Hafner and Lyon 1996).

The basic architecture was simple in design, but complex in reality. Because each computer system had its own unique software and hardware systems, a generalized interface needed to be designed to allow each system to connect to the network. The solution was a dedicated system of Interface Message Processors (IMPs or the equivalent of today's servers) that had the dedicated role of handling connections and managing information flows within the network. In essence, the net was a dedicated communications loop (like a ring road around a city) with each computing center having a dedicated link (on ramp). This basic architecture continues to structure all network designs no matter how large or small. The other basic but crucial design choice included designing the network as a packet-switching architecture (Hafner and Lyon 1996). The basic concept being to break down information into pieces (or packets) that can be individually transmitted across the network. To continue the metaphor above, packets are like cars on the ring road, individuals broken up into multiple units, rather than consolidated into one vehicle and route such as on a train.

The network faced two crucial problems that packet switching solved. First, without packets, transmission between two points on the network would have to be on dedicated lines. In this way, if UCLA and UCSB were using the network any other users would be locked out until finished. This is much the same as when two people are using a telephone. Even if no information is being transmitted, the circuit is dedicated to that specific connection. This was contradictory to the very notion of creating a flexible, inexpensive and efficient means of connecting multiple systems. Packets allowed for multiple users to transmit information simultaneously, with packets from different users flowing through the network to be reassembled at their destination. “Dead” or empty spaces in transmission from one system could be filled by packets from another, creating a constant flow of information in the network.

The second problem focused on the reliability of the network. Loss of data or the cancellation of connections presented a huge problem for a network seeking reliability and robustness. The solution, again made possible by the concept of packets, was the “store and forward” concept. The IMPs would receive packets of information, store them, then forward them only when the connection was open and previous packets in the same sequence had been received correctly. This enabled the network to be incredibly flexible and robust. Each individual packet could be routed on the most efficient path to its destination, and if packets could not be delivered, they would be held until the receiving system was ready.

This basic structure, and its initial success, established the overall framework in which all networks have been developed. The lessons and innovations in this basic design have merged with the software and PC trends outlined above to move networking to an ever-greater number of users and applications.

Two of the most significant breakthroughs for the expansion of networking were the creation of Ethernet and the TCP/IC protocol (Moschovitis et al 1999), both algorithms for managing information flows within and between networks. The Ethernet was invented by Bob Metcalfe at Xerox PARC in 1973. The revolutionary aspect of Ethernet, much like the ARPAnet and ALOHAnet that it modeled, was the ability to create local area networks (LAN) where multiple users could simultaneously access resources from different locations. Ethernet signified the end of time-share computing as a model for organizing computer resources, and assured that the nodes in the ARPAnet could be extended via *intra*nets to multiple users at each University. In other words, individuals could for the first time access a network of computing power, both locally (via the Ethernet) and nationally (via a specific node's connection to the ARPAnet).

The second revolutionary transformation was the concept of connecting networks themselves, that is creating *inter*nets that connected individual networks into workable wholes. The basic features of transmission control protocol (TCP) were outlined by Vint Cerf and Bob Kahn in 1974. TCP simply was the creation of the architecture and language crucial to building a “network of networks”. The basic function of TCP is that it establishes a common language at “gateways” between networks through which information can be exchanged and forwarded to their ultimate destination. In 1974, the power of TCP was demonstrated by the successful sending of a message between the three technically distinct packet-radio network, satellite network and the ARPAnet. The successful completion of the transmission outlined the feasibility of internetworks, and was made even more robust with the addition of an internet protocol (the IP in TCP/IP) in 1978 to handle the routing of messages over multiple networks. This opened for the first time the possibility of any user to access the computing power of any network in the world.

These basic developments created the environment in which, as PCs became ever more present, individuals increasingly began to share information and communicate with ever-wider audiences of users. By the end of the 1980’s, the vast majority of scientists and researchers, as well as many corporate users in the US and around the world were active users of both intra- and inter- nets. ARPAnet spawned numerous other networks for either specific military or science use, as well as the creation of private commercial networks (Guice 1997, Hafner and Lyon 1996). Contrary to the original intention of the network, the main push behind the interest in the internet was the use of email programs, and not the exchange of research or sharing of computer resources. The transformation of the internet into a generalized global and mass phenomenon occurred with the creation of HTTP and the Internet "browser" in the early 1990's.

Up until the creation of hypertext and internet addressing, sharing information through the networks was complicated by the lack of a common language for information exchange. The variety of computer systems, while having a common language in TCP in which to exchange information or commands, prevented the actual reading of information by the receiving computer if some incompatibility existed between the two systems. Just as what happens when someone sends a WordPerfect document to a computer that only has Microsoft Word. In early 1991, Tim Berners-Lee and a group of CERN programmers created three essential tools to overcome these limitations: Hypertext Transfer Protocol (HTTP) that allows for a common platform for the exchange of digital information between computer systems, Hypertext Markup Language (HTML) as the basic programming language for creating documents for HTTP exchange, and the Universal Resource Locator (URL) that allowed for universal internet addresses to be assigned to specific information locations. They called their system the World Wide Web (Moschovitis et al 1999). This established the framework in which an individual could connect to any computer network in the world and communicate in an identical programming language that would allow for the exchange of information.

The architecture existed for an expanded use of the internet, but the actual opening up of the network depended on creating a more accessible interface for individual computer users different than the command driven text model that dominated computing at the time. The creation of the Mosaic browser software interface, by Marc Andreessen and Eric Bina at the University of Illinois in 1993, gave networks and internets a GUI interface similar to what people had come to expect in their PCs. This was the final piece of the puzzle that caused the hyper-expansion of the "Internet" as it exists today. Finally, 24 years after the launch of the ARPAnet, individuals had the ability to connect to a network of computers, create and exchange text and graphical information, all through a simple graphical interface. And just like the early networks, email (in addition to religion and sex) helped propel the network from a relatively closed network of scientists and researchers into a mass phenomenon and industry in its own right.

viii Post-PC Era

The completion of the pieces of the “net” puzzle cemented the final end of computing as it has been structured to the present. From a longer historical point of view, the basic pattern of ever decreasing computer size with increasing connectivity and ease of use has been a constant in the industry's development from Babbage’s time to the present. However, computers, that is the basic box through which we manipulate information, are no longer stand-alone machines. Like the original goal behind the ARPAnet, the network is now the computer. This is not to argue that PCs will quickly fade from existence, but rather to recognize that both the long term and short term trends keep moving technology away from stand-alone computing towards information handling and manipulation within a networked environment.

This movement has followed the embedding of ICs in ever-greater quantities and an every wider number of products. Wireless computer based networking is already a reality, but wireless appliance driven connectivity is just unfolding. The increasing blurring of computation and the natural sciences presents the possibility of universal connectivity of both biological and created systems. There is extensive research trying to embed binary code on DNA, creating biological computational structures. This has been paralleled by attempts to use basic atomic structures as binary circuits (Wired, August 2000). Both of these are based in an increasing move towards nanotechnology, that is cell or atomic level creation of “information machines.” These moves signify the increasing blurring of once distinct disciplines and definitions of IT. TCP/IP, from its very first test, has been designed to connect multiple and differential networks. Over the past decade, TCP/IP has been slowly extended to ever more devices, ranging from cell phones to kitchen appliances (Nijhawan 2000). The new bio-computing models will come with such connectivity built in. This means that IT will increasingly be embedded networked computing, with an ever-greater range and complexity of access and connection linking natural and built environments.

ix Software is IT: The Bricklayers of the Networked Society

The basic patterns outlined in the historic development of information technology still structure the industry today. Beginning with the insight into binary systems and algorithmic logic, which combined with the desire to have machines perform tasks too complex or too critical for humans, IT has developed into a basic medium for the production, manipulation and dissemination of information in all forms. This is paralleled by a tremendous push to digitalize human knowledge, that is place it in binary and algorithmic form, in order to adapt it to this basic IT structure. The desire for flexibility that has characterized the development of IT from the beginning is most easily achieved through software tools and methods. The evolution of IT to the embedded networked computing structure developing now is based on the layering of level upon level of algorithms and digital information, some hardwired and some soft. What is essential to signal, however, is that the process of both building algorithms and digitalizing information is increasingly coming to be dominated by software rather than hardware structures[7]. This is clearly seen in the history above, but it is perhaps little understood because of the relative recent development of software as a stand-alone industry, product and process. Not only are computers increasingly dependent on software, but the Internet itself — with algorithms embedded in software form for TCP/IP, web browsers, network routing and management, Java scripts, databases and commerce to name only a few — is a reflection of the dominance of software as the central method for digitalizing algorithms.

Summarizing the trends that have shaped the development of IT over the last 300 years, IT can be defined as:

- The expansion of binary logic and architectures to dominate all IT forms, giving rise to the mathematical based digital mediums of information exchange.
- The slow transformation of information from analog to digital and algorithmic forms.
- The increasing dominance of algorithms to structure all aspects of information technology, from hardware to software to biology to media. The history of IT, especially for the last sixty years has been the layering of algorithm upon algorithm to create complex interconnected information networks and machines.
- A constant miniaturization of processing power into smaller and smaller components.
- The increasing dominance of software over hardware as the optimal solution for understanding, developing and implementing information technology.
- The pervasive interconnectedness of all IT systems, increasingly wireless and embedded in both organic and inorganic models.
- The public dissemination of intellectual property linked to the implementation of this overall architecture.

It seems extremely probably that these basic patterns will structure the development of IT into the future. The widespread existence of algorithms in social and natural processes creates a huge catalog of patterns that form a natural bridge between IT and society. This bridge is increasingly being crossed,

[7] Hardware, like software, is structured around algorithmic codes that determine the basic circuitry of an IC. In this way, hardware and software both follow the basic structures of binary, algorithmic logic. The real choice is between whether such patterns are "hardwired", with linked tradeoffs between flexibility, reliability, adaptability and cost. Software increasingly exceeds hardware in each of these areas.

and the lines being blurred, between what is beyond the reach of IT to understand, and thus transform and manipulate (Kurzweil 1999). This means that software, and the logics embedded within them, will continue to dominate IT into the future. In essence, if one aspect of the global environment is connectivity through IT, then software are the bricks and programmers the bricklayers of the networked society. As such, understanding how social processes are defined and transformed into digital logics, that is how the architecture of IT is defined and built, becomes essential to understanding the structures and impacts of IT in the economy and society.

IV Software as Process, Product and Industry: Domain Knowledge, Embedded Information and Softwareism

If software is the brick and software programmers the bricklayers than software code is the raw material of information technology. Code is the basic language in which processes are translated into the binary code that can be processed by computers. The structure of creating software code follows a very basic pattern. Any process, such as typing words on a page or the interaction of two computers over the internet, is defined in high-level language, that is on a macro-level of the basic operational procedures and outcomes. This is then detailed in a low-level design, a design that outlines the specific structure of the process. This design is then translated into the actual code by a programmer. At either the low-level design or coding phase the actual process is placed into an algorithmic form, that is the logical mathematical sequence of events that will produce a desired outcome. Below is one of the most common examples of code at this level, the initial header from an HTML document, that is the basic code interpreted by a web browser to specify how text and images should appear on the computer.

```
<html>

    <head>
        <meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
        <meta name="generator" content="Adobe GoLive 4">
        <title>Center for Global, International &amp; Regional Studies</title>
        <script language="JavaScript" name="pop_up_once">
```

This code is then compiled, meaning translated from one of any number of programming languages that humans understand into “machine code” or the binary form composed of zeros and ones (e.g. 0010100011) that computers can understand. At this point, this is software and no longer code. It is an actual program that can be replicated and distributed as a complete product. It is entirely possible, in fact almost guaranteed, that the software will not produce the desired results as specified in the initial high-level design. Typing “A” may produce “a” or even worse “Z”, computers may be able to connect but unable to transfer data (Stephenson 1999). Such problems are commonly referred to as bugs, and the last stage of software development is removing these problems before the product is finally released. Bugs, however, can stem from a number of problems along the whole development process, from basic flaws in the high-level design, mis-specifications in the low-level design, faulty algorithms, or simply mis-coding or even just mis-typing (Kohanski 1998). As such “debugging” can be a time consuming and demanding process that is a combination of luck, skill, experience and sheer effort. Once debugged, however, the software is then recompiled and finally complete.

What is important to clarify is that the algorithmic logic and the software development process structure how information becomes digitalized (Negroponte 1995, Mitchell 1998). The coding of algorithms, whether in a fixed form as in hardware or flexible as in software, is an essential feature of IT. Software, because of its flexibility and role in structuring the layers of protocols that shape the Internet, is increasingly the chosen form for the application of coded knowledge. In the case of algorithmic logic, software is essentially embodied social knowledge. Algorithms and thus software can only model those processes that are well enough understood to be translated into a logical sequence of procedures and outcomes. The greater the translation from analog to digital procedures, the greater the knowledge embedded in software and information technology. In this way, it has been argued that software is not a product to be produced, but a process of storing knowledge. Software – like DNA, the human brain, hardware and books – is just the most recent medium for knowledge storage (Armour, 2000).

However, the unique feature of software, as compared to these other mediums of knowledge is its persistency, update speed, deliberate flexibility, and applicability to action (Armour, 2000). Software has each of these qualities that are either absent or less dynamic in the case of the other mediums. More importantly, and perhaps the greatest strength of software, is its ability to emulate each of these other mediums of knowledge. For example, software can emulate the best features of books, and is increasingly replacing hardware, especially at the IC level, in many cases[8]. It is this basic structure that gives software its unique role in IT. It is also one of its central limitations. Software can only emulate

[8] The case of Transmeta, which has created a software structured IC designed specifically for mobile application, is an ideal example of this (Wired, July 2000).

what is understood in algorithmic form. The increasing demands made of software as the processes that it must model increase in complexity limit both its reliability and application.

The other central limit of software, and thus information technology, is the very process of software development itself. The process of software development outlined above is entirely accurate. However the essential feature that is not conveyed is that this development process strongly resists rationalization. The natural process of developing software, precisely because it embodies unique processes and knowledge, tends to be arbitrary and crafted, rather than defined and manufactured (Raymond 1999, Stephenson 1999, Kohanski 1998). If two processes are identical than the exact same software program will suffice for both. However, the slightest difference in structure or desired outcome can have a dramatic impact on both the software design and the algorithm linked to it. Even more essential is that there may actually be more than one algorithm for producing a desired outcome. In some cases there are defined answers to questions of efficiency and structure, but in others it is extremely difficult to rank the differences in logical approaches. And what determines these differences in logical approach to begin with? Do culture or social structure matter, and in what way and to what degree?

Even more challenging is that many of the processes being emulated, such as DNA or ecosystems, are beyond the ability of humans to fully understand, ironically, even with the help of computerized information machines. This lack of basic understanding is complicated by the structure of software design that in itself creates errors and mistakes. As such, software is at the center of IT, which increasingly is the essential medium for most forms of information flow and manipulation, from natural to social to economic structures. However, software is also inherently a social product – it is limited and defined by both the social models that determine its algorithms as well as the inherently craft-structure of the software development process itself (Agre and Schuler 1997).

Overall, software processes and architectures will increasingly influence the development and organization of many of the basic social structures being constructed within an information dominated society as they become codified into digital forms. In this way, “code is law” (Williams 1998), that is the fundamental pattern that structures spatial and temporal, and thus social and economic, interactions. The basic development structure, combined with the congealed knowledge inherent in software, distinctly influence the development of the IT industry, and IT is increasingly influencing industry globalization patterns, debates over intellectual property, proprietary versus open-source development models, basic public safety issues, and regional development strategies (Lessig 1999). Understanding software as both embodied social knowledge (Agre and Schuler 1997) and as a fundamental production process within society, directly helps frame an understanding of what the social and economic implications of IT may be. Such an understanding requires considering software as a process, product and industry.

i Defining Software Development

a Software Development Processes

Software has a fundamentally different development process than that of high-technology hardware processes in particular or industrial manufacturing processes in general. Software is essentially a pure digital product. Operating under different production economies than manufacturing, the entire software production process, and virtually all the value added, occurs prior to any final production, assembly or packaging. The value-added comes directly from developing the algorithms that model real processes, and not as commonly thought translating these algorithms into code. There are, theoretically, no overriding geographic or economic constraints (transportation, capital or natural resources) on its production or location.

Metaphorically it is helpful, if limited, to think through software development processes in terms of an academic model. Universities agglomerate highly trained professionals around specific "product" areas producing unique works that are often highly personalized even though they are constructed within broad disciplinary parameters. Individual "products" are designed, created and evaluated by the same set of individuals throughout the process. The vast majority of the value-added is located in this development process, with minor value-added contained in publishing and distribution. This development process theoretically can take place in any geographic location. However, communication between producers is easier within a specific geographic location, usually within the same institution. The physical closeness also produces an environment of "innovation" or "discovery", where ideas are quickly critiqued, shared and blended together. Furthermore, increasing the number of producers increases the need for communication and often lengthens the development time considerably, especially as the spatial separation between producers increases. The final product is distributed on the academic "market", where it is often harshly evaluated for having fundamental flaws that should have been corrected through better design or writing or pre-shipment evaluation. This often instigates a revised version of the initial product that incorporates new features while repairing old flaws.

However in academia and software alike, any problem or area of focus is subject to myriad forms of analysis and interpretation depending on an individual academic's training, perceptions, beliefs, environment and incentives, even if the general conclusions reached are similar across divisions. Evaluation of work product and individual performance is extremely complex mixing both quantitative and qualitative metrics, and generalized criterion are difficult to determine. Quantity, quality, style and logic are open to constant interpretation and emphasis depending on the nature of the problem considered. Experience and individual performance over time is often far more important than training or environment in determining quality work. At the same time, though quality clearly differs between academics and universities, the general cost of training an academic or creating a university is roughly equivalent. Automation and rationalization of the process are fundamentally impossible given the complexities of the problems considered and the variation in performance of the producers. In times of increasing demand, production is essentially increased through the expansion of the system extensively by creating new universities and academics. If local markets are unable to provide the needed labor force at the expected quality, international markets and networks are tapped to increase supply.

This analysis, while stylized, characterizes the general nature of software development well. Software, like academics, is a labor intensive, highly skilled interpretive field where credentials only partial signal quality work and future capability. Software is often difficult to develop, especially as the ideas being modeled increase in complexity. The difficulties increase as both the number of producers and distance increases (Brooks 1995). Because of this, software is often incomplete, unsatisfactory or filled with flaws. Many software projects, up to 80% are never even completed (Hoch et al 2000), but remain half finished, never moving beyond abstract design. Even completed projects are inherently structured by the domain knowledge and biases of their chief designers. This unique development process in turn structures the geographic and organizational forms that both the software industry and academia take.

Overall, this means that software is characterized by two key factors: the demand for skilled labor and communication at all levels of the production process. First, programming, referring generally to the broad process of designing, coding and testing software programs, is a labor-intensive process requiring specialized skills. Labor must be specifically trained in languages (i.e. HTML, Java, C++, Perl) and concepts of software development (i.e. software architectures, mathematical structures). Given these unique prerequisites, other engineering skills can not be easily transferred to software production in time of increasing demand. While it is possible to use the term software engineering, which I will use to refer to the overall multistage process of software development, it is not engineering in the classical sense. Programmers do use rules of syntax and semantics, but like all languages these vary over time and space. There are no universal physical laws to define all software development. This also means that it is difficult to quantify software processes and products (and thus people) against a universal standard of quality and efficiency. Software, invariably, is a very human centered, almost artistic, process. It is best described as a craft, and Gilb (1996) uses the term “softcrafters” to describe what software professionals do. Like other forms of knowledge work that exist within dynamic environments, the best software development is a skilled process that invariably requires contingent decision making and tacit knowledge that is by definition unquantifiable. This is even more extreme in the case of software exactly because the models being created are from diverse fields that often model social, not natural, processes.

Second, the basic complexity of software design leads to a high demand for communication at all levels of the development process. The need for communication is broader than merely having adequate telecommunications infrastructure. The mapping of real world processes into algorithmic patterns that contain all possible contingencies is fundamentally structured by the assumptions that each designer, programmer and end-user bring to a software product. This means that communication between end-users and programmers must be both highly detailed and interactive at all levels of the production process. The need for communication also increases with the complexity of the system being modeled, as more and more contingencies are built into the system.

As complex systems are broken up into manageable production units with greater numbers of programmers, the lines of communication between each individual unit and programmer increases exponentially (Brooks1974). This is one of the fundamental contradictions of software development. Larger, more complex software products require corresponding increases in the amount of direct labor needed for their production. Yet, the increasing number of individuals involved in a project (and the increasing number of unique assumptions brought by them) actually decreases the efficiency of software production as the lines of communication increase. These complexities are solved through constant interaction between designers, programmers and end-users. It should be obvious that software development is at its most coherent when each of these elements is combined within a single person. As software products increase in size and complexity, from individual to team to firm, the communications transaction costs increase exponentially.

These two aspects, the need for high quality labor and constant interaction, are consistent throughout the software production process: from initial design to coding to testing to integration. This has resulted in geographically and organizationally centralized software production, as firms have sought to capture both interaction and knowledge within organizations. Software production has tended to locate in regions with large pools of qualified labor, or have tended to agglomerate large quantities of skilled labor that then has induced new software firms to locate in that region (Hoch et al 2000). Firms have also tended to keep the entire production process in-house, due to both the proprietary nature of software development and the need for constant interaction over the production life cycle (Cusumano and Selby 1995). To better understand the spatial and organizational structures within the software industry, as well as the constraints on its globalization through a more rationalized division of labor, it is important to analyze the history of attempts to rationalize the software development process.

b Software Rationalization Considered

As outlined above, software development is composed, simplistically, of the following stages: high level design, low level design, code, unit test, system test, beta test (most recently), and final deployment (Jalote 1997). Classic software development processes entailed the involvement of the same set of programmers at each one of these stages for small projects. As projects grew in size and complexity, project leaders would meet with the end-user and design the overall system and then create detailed design specifications. Coding, that is implementation of the specifications, was handed off to teams of programmers that were responsibly for coding and even creating the detailed algorithms, testing the individual units and merging the units for system test. Debugging, and even testing, was a constant process done internally by the programmers themselves. The process was manageable, but not rationalized or systemized. No quantifiable data on time, cost, quality or productivity were available, and as such the process was not repeatable or improveable (Gibbs 1994). Software production has been and continues to be notorious for cost overruns, time to market delays and severe quality problems. All problems related to the craft nature of the production process.

The structural problems of the development process intensified as software was scaled up to meet increasingly complex demands. Problem areas could not be identified and transferred to new projects or other aspects of an existing one. Best practices were locked away in programmers' minds, and as such locked into specific departments or projects in firms and not disseminated to the industry at large. Delays in project completion, tremendous cost overruns (orders of magnitude greater than original budgets), poor quality software and seemingly low productivity (though based only on anecdotal evidence) created intense pressures within the industry as it expanded. By the late 1960's both private and public sector entities began to discuss what became known as the "software crisis". The existing methods of software production were no longer considered adequate to meet the needs of future systems as they grew in complexity and size. In 1968, the NATO Scientific Council convened a meeting of leading software professionals and industry leaders in order to formulate a plan of action to address the crisis (Gibbs 1994). The experts arrived at no agreement on a course of action, but they did create the concept of software engineering as the ultimate goal of any process rationalization.

It is telling that thirty years later, people are still debating the continuing software crisis, and that software engineering is at best a course, but never a degree program, at universities throughout the world. Software development continues to be a "labor-intensive, intellectually complex, and costly activity in which good management and communication count for much more than technology" (Fenton et al 1994:6). Software is still expensive to develop, usually late and often unreliable and subject to constant change and rework. By the mid-1980's, software was considered to be 80% of the cost of new computer systems (OECD 1998a). Seventy-five percent of that cost was not development but maintenance (that is the debugging, upgrades and integration with other software products) of the software after installation (Jalote 1997:3).

Software development has continually resisted the creation of a repeatable, quantifiable and improvable process. A series of "magic bullets" have been presented over the years that have promised to radically alter the nature of the production process. None have succeeded (Brooks1987). Structured design and programming have been attempted. Process models like the Waterfall (Pressman:1997), Spiral (Boehm:1997), Software Life-Cycle (Davis 1997), personal process model (Ferguson et al:1997) and open source (Raymond:1998) have been implemented. New programming languages such as C++ that are based in object oriented programming have been used to maximize software reuse and maintenance. Software metrics have been created that attempt to quantify and map the software process based on TQM principles (Grady:1994). Cost and quality models such as COCOMO (Legg:1997) and the Capability Maturity Model (Herbsleb et al:1997) have been developed. No perfect or even adequate solution to the basic concerns of increasing product complexity, low quality and weak productivity growth has been found.[9]

The most successful software development models over the last thirty years have essentially dealt with the demands on software production through an emphasis on extensive rather intensive growth (Brooks 1995, Cusumano and Selby 1995, Raymond 1999), that is through large increases in the number of software professionals rather than dramatic increases in process productivity through rationalization. Both the structure of the industry and the increasing demand for software labor reflect this process. Firms throughout the economy have outsourced the full software development process in order to capture expertise and scale that were impossible to provide at the needed cost and quality levels internally. This has given rise to software service firms and packaged software products that have sought to maximize scarce skills and products by providing re-usable solutions across sectors of the economy. Such trends have failed to match the increase in demand for software, and merely shifted the need for labor to a different sector of the software industry.

This has also not fundamentally changed the need for constant communication at all levels of the software process, especially as the scale and complexity of software products have increased. The open source movement – chiefly around the Linux OS, Apache and SendMail programs but expanding to other systems – relies on distributed networks of programmers who write and debug code based on ties of community and ideology and not financial compensation (Raymond 1999).[10] Software development has also moved to the standard practice of beta-testing, the shipping of trial versions of software to groups of users, usually at no cost, prior to final product shipment. One of Microsoft's chief strengths,

[9] A short example of the weakness of rationalization attempts is instructive. The capability maturity model is a five stage program designed to assess an organization's software development process. Few organizations have gone through the expensive and time-consuming process (a process that often takes months and an organization's full attention and resources) of evaluation and ranking. Fewer still have attained the highest level, level five ("optimized"). However, it is important to note that level three in the rankings ("defined") is equivalent to ISO 9000 standards, which many firms have actually accomplished. Models such as the CMM have consistently failed both because they fail to understand the nature of competition under current conditions and are theoretically, not practically or experimentally, derived. Perhaps more importantly, the individual and diverse nature of software development makes such standardized solutions at worst widely inapplicable throughout the industry, or at best only partially successful for a limited number of firms producing specific products (i.e. military contractors producing complex, mission-critical systems), where cost is not a concern.

[10] The open source movement provides an interesting organizational counterpoint to the geographic and firm agglomerations that have characterized the for-profit sector of the industry. It is clearly one, if not the only development,

and one that compensates for the firm's weaknesses in the software development process, is its extensive network of beta-testers. Windows 95 was shipped to 400,000 beta-testers prior to final release. Such testing has been known to detect 25% of all bugs in a system (Cusumano and Selby:309-313). However, as stated above, such extensive solutions generally only increase the complexity of the development process with no relative increases in productivity or quality given the increases in complexity and size of software products. Even the most celebrated success stories, specifically Microsoft and the open-source movement, rely on large amounts of unpaid labor to overcome the fundamental inefficiencies and bottlenecks of the software process itself. As such, calculating the true costs and productivity of software development is a much more complex task than traditional economic analysis would imply.

Overall, there have been incremental improvements in software products and processes over the last thirty years. However, it is important to understand that these improvements have in large measure been extensive, that is labor intensive based, rather than through productivity increases derived from process rationalization. Furthermore, even these improvements have failed to keep pace with the complexity and size of the software programs demanded by users. A demand that has been exacerbated by the increasing embedding of IT in an ever greater aspects of society, entailing the expansion of software into entirely new areas of domain-knowledge. Improvements in process quality, productivity and management have failed to match the increasing demand, size and complexity for both customized and packaged software. At the same time, the full utilization of the existing software labor pool within regions and nations, most importantly the United States, has pushed a spatial and organizational reorganization of the industry as it searches for new sources of labor (Hoch et al 2000).

ii The Impact of Software and Software Practice in an Information Society

To understand the impact of software and software development models, it is essential to demonstrate the role of software within the global environment. There are three ways to measure the impact of software: the size of the global market and industry, dependence on software driven systems, and the creation of unique practices that would be impossible without software dominated IT (Hoch et al 2000).

that could alter the direction and structure of the industry in the near future. However, open-source software still does not solve the fundamental problem of an extensive based development process that limits a quantifiable, rational process.

a The Size of the Global Market

Software is representative of new products within the information economy that are difficult to classify using measurement tools derived from a national based industrial economy. The existence of large custom projects, in-house software development, differences in national accounts and definitions of software, confusion in defining software as a service or product, the development of software outside the formal IT industry, and global software labor processes have all made accurately describing the industry development difficult.[11] However, a rough consensus of the global industry and market can be drawn as follows:

- In 1998, the global market for software products and services was approximately 470 billion dollars.
- Of this total, customized software was 41.5%, prepackaged software was 36.2%, integrated system design was 12.3%, and 10% was software services.
- Since 1985, sales have been growing at 15.9% CAGR.
- The global market is expected to exceed 1.7 trillion dollars by 2008.
- The US share of the global market and production is approximately 50%.
- In 1998, world direct software employment was estimated at 2 million.
- World direct software employment is forecast to grow to approximately 3.6 million by 2008.
- In the United States, direct software employment was 806,900 people in 1998.
- United States direct employment is forecast to grow to 1.3 million by 2008.
- In the United States, 72% of software workers are employed outside the formal software sector.
- By 1995, software packages and services had exceeded the value of hardware within the IT market.

(National Software Alliance 1998, Business Software Alliance 1999, OECD 1998a)

The US will continue to form the center of global software production and sales well into the next decade. Customized software products and services, while decreasing from a 100% share of the market in the early 1960's, will continue to be important with a decreasing trend. Important sectors will most likely focus on customized services rather than software products. In other words, maintenance, repair and customization of legacy and packaged systems will form the majority of work in this sector, as the Y2K experience showed and as legacy systems are transferred to web compatible architectures. Standardized solutions to general problems of network connectivity, content provision/interaction and organizational productivity will increasingly come to focus on standardized software solutions that can be customized to meet end-user needs.

[11] The best example of this is software shipped internationally via the internet or on CD-ROM. The first transaction, if captured at all by national accounting, is usually considered a local service and usually not taxed. The second is considered a manufactured product with tax usually paid on the CD-ROM itself, but not the value of the software. Anecdotal examples of the confusion brought by using industrial methods to measure a digital economy abound.

In terms of the impact of software in the global economy, the most significant trend is the number of software workers outside of the formally defined software sector. This indirect employment indicates both the importance of software within the economy in general and the inability of traditional market numbers to accurately reflect the true ubiquity of software development processes in all sectors of the economy. In the United States, and not included in the direct employment totals from above, are over two million individuals employed in software production and services outside of the software industry (BSA 1999). This places global direct and indirect software employment at approximately 7 million in 1998, growing to an estimated 12.8 million by 2008.

b Software Dependent Systems

Software is increasingly replacing hardware in many mechanical and computer systems, either directly or as firmware, that is reprogrammable ICs. The flexibility that software offers in terms of upgrading or changing systems increasingly places software in individual systems like airplanes, automobiles, factories, basic appliances and consumer goods, as well as complex systems like telecommunications, financial markets, energy generation and transmission, supply logistics and national defense. The failings of software systems have almost reached the point of urban myth, but in fact are very real. The intense concern generated by Y2K, the Euro conversion process, the “I Love You” virus, as well as notable software failures at new airports in Hong Kong, Kuala Lumpur and Denver have shown the increasing embeddedness of software, and the dependence on these systems, in all aspects of society (Ullman 1999).

A useful example of this, and one that highlights the increasing influence of software on hardware processes, is the distinction to be made between designing (as an electrical engineer would do) and programming (as a software engineer would do) (Gibilisco 1997). Classically, integrated circuit engineers would design electronic circuits based on sound principles of engineering and physics. Increasingly, however, IC design uses software and specific programming languages to code and simulate IC designs and test them virtually (SCIP:1999). This basic pattern of software based design and testing is increasingly common in most industrial sectors with the extension of CAD and CAM systems, including consumer products, automobiles, and mineral exploration. Transmeta has taken this a step further, creating a standard IC that operates within a software shell that interacts with the overall computer environment. This means that not only design, but also the actual operation of the IC is software based and dependent (San Jose Mercury News, August 8th, 2000). This gives Crusoe, the Transmeta chip, the versatility and flexibility of software combined with the speed and robustness of hardwired code (Wired, July 2000).

c Unique Software Based Practices

Clearly, there is a growing dependence on computer systems in general, and software in particular. This is at the heart of the history that spurred IT development. However, software is increasingly at the heart of scientific, commercial and media processes that would be impossible to replicate without software enhanced tools. The recent mapping of the human genome is also an excellent example. The success of Celera in completing the DNA outline prior to the government led Human Genome Project rests on the proprietary algorithms, in the form of software run on a series of supercomputers, developed by the firm to determine the exact sequencing of the identified individual gene segments (Ridley 1999). Another example is the effort of the US government to promote a ban on nuclear testing based in large part on the ability to replicate such explosions in computer generated environments. Perhaps the simplest and most direct example of software based system is the Internet itself, which increasingly shapes commercial (B2B processes), social (geographically distributed communities) and media (virtual worlds) processes that would be impossible to replicate without software. As outlined above, the very architecture of the Internet as a medium that can merge multiple technologies and tasks would cease to exist without software.

Overall then, measured by the impact on the global economy, dependent systems and increasingly unique practices, software is a central factor in the global environment. It also means that the inherent limits in its development process, that is the coding of human knowledge into algorithmic forms, also will increasingly shape social and economic processes. A programmer could design a program that is algorithmically sound (based on the parameters the designer chooses) but which would fail when applied in a real world situation. The Y2K bug is a classic example of this, where assumptions of the direction and pace of technology change became embedded in the architecture of the system. Software designs require the mapping of complex and evolving systems based on a programmer's understanding and assumptions about those processes. No formal and invariable laws (like the laws of physics in engineering) exist.

As markets merge within information technologies, the lines between software and other industries will most likely become more blurred. The constant expansion of embedded systems, tied to the push for ubiquitous computing and connectivity, will challenge our ability to understand the importance of software in the economy, as such processes will be come ever more hidden. However, while the actual economic impact may become more hidden within other sectors of the economy, software development processes and the problems stemming from its limitations will become increasingly more prevalent.

iii Softwareism (!?): A Working Hypothesis on the Impact of Software in the Global Environment

With the centrality of software in the global environment, it is important to analyze the potential impact of software in structuring organizational and institutional policies and practices. Such an analysis needs to consider both how software products structure social practices (through the knowledge they embody) and how software development processes frame economic activity (in terms of how knowledge is transformed into products). In the first instance, how social practices and knowledge are codified into software becomes a central issue for understanding social and economic policies. Software, or IT generally, is not only an architecture that facilitates global flows (Castells 1994), but actually structures how such flows occur. Assumptions about social processes are inherently built into the algorithms that comprise software. As software increases in importance in society, such assumptions increasingly impact upon how social relations are structured. This in turn suggests that software increasingly may shape the strategic and organizational choices of governments, firms and civil society, whether conscious or not (Lessig 1999, Williams 1998).

Returning to the example of human genome project above, it is enlightening to elaborate on the chosen business model of Celera. Rather than patenting the individual gene discoveries, as was possible, Celera chose to distribute these results freely via the Internet (Wired, August 2000). Widespread access to the genes was thought to promote the use of Celera's real comparative advantage, the proprietary algorithms for analyzing gene sequences and the accompanying database that contains such sequences. In other words, Celera is not a biological research lab, but a company that converts biological knowledge into binary forms that can be manipulated and analyzed. Its expertise lies in its ability to do such analysis more rapidly and cost effectively then anyone in the world.

At each stage of this brief outline, the impact of IT in structuring the definition, distribution and access to knowledge is apparent. Celera through proprietary algorithms define what gene sequences are relevant. Current understanding of the humane genome consider up to 90% of the genetic code as “junk”, that is meaningless information (Ridley 1999). Such assumptions, true or false, are built into the software that determines which sequences are relevant. The distribution of the information on the Internet structures access to the information itself, both in terms of access itself and the quality of that access. And what Celera specializes in, its competitive advantage, is not the mechanics of research, but in the definition of how such research should be structured and then coding such research in a proprietary software product. Examples such as this raise issues concerning not only the direction of information technology (as articulated by Bill Joy and Lovins, Wired, August 2000), but also how IT frames how scientific and social questions and answers are structured. Looking at software in this manner opens up considerations of software products as embodying norms and regulations (Lessig 1999) that regulate social interactions.

Looking at software from this first aspect leads into considerations of the second important facet of software as a production process. Because software is increasingly a central method for producing, storing, transforming and distributing knowledge, how such processes occur frames how social knowledge becomes embodied in actual products. In other words, software practices define how information is produced and translated into power and profit on the “shop floor” (or cubicle) level. From the discussion of software development processes above, it should be clear that regulating software production is a very different proposition than regulating industrial production. Because software comprises a very tacit, unqantified process, regulating both the inputs and outputs of that process are extremely challenging. In many ways, software production has an artistic and “free speech” quality that makes regulation almost non-sensical. How can two algorithms be compared, and what is a quality software product? Answers to such questions are inherently limited by the software development process. Algorithms work on assumptions of how processes occur, and algorithms are created through very distinct processes that are socially structured.

As suggested above, the software industry, whether consciously or not, recognizes this essential characteristic of software. The organizational patterns of firms and the structuring of software production reflects an understanding of the social basis of innovation and production. In other words, the vertically integrated, product centered firm — linked first to global networks of immigration and then to distinct regional centers of software development globally — make organizational and strategic sense when viewed through how software is produced. The increasing application of software solutions to new areas of the economy and society imply that both the production processes and organizational structures that support them will increasingly have a broad impact in the global economy. Focusing on the software industry as the leading example of these new production structures should also not confuse the fact that such processes, based in part on the increasing importance of software in general, are becoming more dominant in the IT industry overall.

The dominance of software production processes and organizational logics helps explain or elaborate on many key concepts within theories of globalization and the information society. At the most simple level, analyzing software practices help explain how processes are both global (a global market for software products and labor) and extremely regional (a need for local knowledge and interaction) simultaneously, and how such relationships are structured through global networks and flows. In other words, understanding software production processes may help link micro-level (shop floor) processes to broader organizational and institutional trends in the global environment.

In this way, understanding software processes and products implies an understanding of the informational aspect of IT specifically and the "information economy" generally. While this may seem overreaching, it is an essential consideration if many of the theories of globalization, the networked economy, flexible specialization and regionalism considered above are to be evaluated in a concrete and predictive manner. The extensive literature that considers Fordism and Taylorism indicative of essential social relationships, production processes and organizational structures in the twentieth century (Doray 1988, Giordano 1992, Waring 1991) establishes a pattern for linking "shop-floor" and larger macroeconomic and social trends. Debates on the transformation of this structure, linked specifically to "Japanese-style" production systems (Jurgens et al 1993, Sheldrake 1996, Womack et al 1991), suggest that such micro-level changes are occurring and impacting upon broader industry and social patterns. As such, if the impact of software is as widespread and the production process as unique as suggested above, then it is possible that "softwareism" may define the dominant mode for organizing and producing social and economic goods in an information society.

Similar arguments have already been made for IT, particularly the semiconductor industry. In arguing for an analysis of IT as a "new mode of industrialisation", Henderson considers that "semiconductors are not only the heart of microelectronics and information industries generally, but that semiconductor companies themselves constitute a production and organizational form that is a paradigm example for the global option in practice (1989:4)". Angel (1994) continues this line of reasoning by stating that semiconductor firms are optimized for innovation, which in turn shapes the organizational and geographic structure of the industry. This "structuring for innovation" is embedded in the linking of flexibility and innovation on the factory floor, shaping formal and informal learning and information exchange.

However, while interesting analytically and methodologically, such analysis is far more similar to the considerations of industrial practices under Fordism or Taylorism than IT. There is no real clarification of what comprises innovation, knowledge or information, and how such practices structure an "informational mode of development" (Castells 1989). Equivalency to the software industry example outlined above would require that semiconductor case studies focus on the globalization and organization of semiconductor design and prototyping alone. This would provide a clear example of knowledge or information production practices, as well as defining the importance of such processes (in terms of value added or productivity) for the industry overall. Such an analysis would help clarify if a "new mode of industrialisation" or simply a modification of existing industrial manufacturing forms was occurring, and what the social and economic consequences stemming from such changes might be.

If an information logic is dominant in the global environment, a preliminary hypothesis is that looking at semiconductors or other industries in this manner would reveal similar patterns to software development processes and organizational forms. As such, "softwareism" can be defined as:

- ➢ A process that is organized around the definition, generation, manipulation and transmission of information into socially and economically applicable forms.
- ➢ Because it is socially structured and often determined, the production of such congealed knowledge will often take the form of craft-like (or creative or research-like) production systems where tacit (as opposed to explicit knowledge) is essential, human resources are the central resource, peer or market review processes dominate, the division of labor is weakly defined and production processes are non-rationalized.
- ➢ Increasingly value-added will be greater in the design or mapping the algorithmic aspects of a process, that is in the ability to define and model a process than in its actual implementation, manufacture or replication.
- ➢ Flexible networked organizational forms that are able to efficiently and rapidly manage the flows of information (both its creation and applications) and manage "information or knowledge

workers", and implement the new tools for its manipulation and dissemination will take precedence.

- These organizations will locate in regional environments that produce knowledge and the cultures that stimulate its transformation into congealed marketable forms.
- Regions globally networked through flows of individuals will tend to predominate, individuals being the key social institution for transmitting tacit knowledge as well as providing the highest value added within the global environment.
- Labor markets will increasingly be globally defined, though product markets will be fragmented.
- Importantly, the social nature of domain knowledge, that is the place and context specific understanding of processes that underlies the logic of software, insure that regions and culture will play a significant role in the development of IT.

Methodologically, these general components of "softwareism" are the patterns that form an ideal type against which comparisons and insights to other industries can be made. This, as in the case of Fordism, Taylorism or semiconductors, would not mean that all nations, regions or organizations would follow this exact model, but that they would share the common characteristics outlined above or exist within environments where such a framework is operational. A simple heuristic model would suggest that such an environment would be global with strong regional agglomerations formed by a need for skilled workers and domain knowledge. It would contain globally distributed mechanisms for congealing social knowledge, as both a resource and an end product. It would include flexible and dynamic processes linked to the very tacit nature of its production process. And importantly, it would consist of new organizations and institutions to support such an environment that would create new forms of social interaction and social conflict.

Much of the analysis of globalization and the information society (Castells 1997, Gordon 1994b, Held et al 1999, Amin 1994, Drucker 1993) has already established macro patterns similar to this heuristic model for the nature of innovation and knowledge, patterns of global flows, network organizational forms, and new social trends. As such "softwareism" may serve as a clarification of why information is valuable, and how it is structured and valorized in the global economy on a micro-level. It also opens a way to consider how a basic process like the commercialization (or transformation into authority) of information creates strong similarities in production and organizational patterns across industries like software, biogenetics, film, music, academia, media, design, architecture and finance. As such, it may open up a way to understand broad organizational and institutional trends as well as the blending of certain industries and institutions. Most importantly, it may present a method for grounding broader theorizations in detailed empirical work and case studies that can lead to a clearer definition of the forces at play in the global environment, including better government policies and firm strategies to navigate within such a structure.

V IT and Development: Considering State-led Regional Hardware and Software Strategies

As software has increased in importance within the economy, governments have begun to consider policies and strategies for fostering software industries. Contrasting such initiatives with previous hardware strategies helps elaborate on a number of the issues considered above. On one level, by contrasting two sectors within IT, it is possible to clarify how different the two sectors are, and with what implications for development strategies and social structures. On another level, it offers a very exploratory consideration of broader shifts in the global environment. New development strategies focused on software may actually signal a recognition by states that there is shift in the global environment from an industrial to an informational-based economy. In either case, it is important to consider how states structure strategies and respond to challenges derived from IT industries, specifically to help frame future research questions and programs. As such, analyzing regional IT strategies offers a means to compare and contrast regional initiatives individually as well as pull out broader commonalties that reflect more global trends.

Government action is an essential factor in linking regions to broader national and global networks (Ngai-Lim in Amin and Hausner 1997). IT development initiatives have increasingly come to be the domain of local-regions within the global economy (Scott 1996, Storper and Harrison 1990, Barnes and Gertler 1999, Malecki and Oinas 1999). National development policies, whether focused on economic development agendas or defense concerns, have often provided the basic resources and markets that have structured regional development projects. Tax incentives, public infrastructure projects, educational investments, government purchases, R&D funding and legal guarantees have all formed a basic package that multiple national governments have employed to promote regional development. However, these national initiatives have been complemented by regional development strategies formulated by local governments and firms. These strategies have often taken the shape of regional development agencies that have taken advantage of national policy openings to provide focused, centralized and responsive local initiatives promoting development (Lubeck and Eischen

1998). Over the last three decades, the IT industry has increasingly become a central focus of these regional initiatives.

Most recently, regional-states are increasingly focusing on transitioning from IT hardware focused development strategies as software increases in importance in the global economy (Hoch et al 2000, Britcher 1999, Egan 1996). While many of the same economic and social forces that have structured hardware based development strategies over the last few decades apply to software, a software development strategy signals a shift to new strategies of competition for regions in the global economy. Like hardware initiatives, software strategies are by definition specifically created to integrate and operate within global networks of production, immigration, innovation and competition (Karolak 1998). However, these networks are structured by the limits defined by software processes. Unlike hardware, where innovation is separate from production or limited to rational and quantifiable processes, software initiatives must capture within regional centers the tacit processes and resources for innovation directly. Within the software industry, this innovation is increasingly structured around the codification of specific domain knowledge through software processes. As more information becomes digitalized and more processes become software dependent or influenced, development initiatives will need to focus on innovation around products and services that combine world-class software skills with linkages to other industries and domains of knowledge.

Such innovation may be captured in the form of large national firms, but it is much more likely captured in regions of innovation sustained by a skilled-labor force that may or may not be citizens of that country. Software in this way is about the incubation of ideas and skills, rather than the capturing of defined and pre-determined resources. The major economic and organizational benefits to the nation or region derive from having a "milieu of innovation," that is a network of innovative firms and individuals, within an information driven economy, not from generating national firms or large-scale employment (Gordon 1994b). In this way, the software development model will succeed or fail based on investments not in infrastructure per se, but in creating an extensive human network that can sustain a regional culture of innovation (Scott 1996, Saxenian 1994, Storper and Harrison 1990).

Yet, the ability of states to create and direct such innovative knowledge driven development remains only partially understood. Software, much more than hardware, exemplifies the need for "learning by doing, using and interacting (Borrus 1993)". The tacit nature of software technology inherently limits what can be bought or transferred through infrastructure investment in the form of FDI. In other words, software requires much more interaction in its development, use and application because the development process is so undefined and human capital based. The creation of this interaction, on both a global and local level, is central to creating even the most minimally successful software milieu of innovation (Malecki 1997, Malecki and Oinas 1999).

The general characteristics of software focused strategies are further confused by the extensive literature analyzing high-technology development initiatives. Much analysis of IT development initiatives suffers from either mis-stating or not defining exactly what IT is (Evans 1995, Ernst and O'Connor 1992). Such analysis often places production of basic computer components or assembly as equivalent to advanced semiconductor manufacturing or software development. This obscures the policies needed to both initiate and sustain various IT sectors, particularly when they are as different as hardware and software. Other analysis focuses on the production of specific sectors, but fails to distinguish between various production patterns that exist within these specific value-chains, especially the trade-offs between developing regional design or manufacturing capabilities (Ernst 1997, Flamm 1998, Forester 1987, Angel 1994). While there are general patterns that tie IT together, chiefly the coded information structure of both hardware and software processes, neither analysis defines these general patterns or elaborates exactly how IT macro-trends link to specific regional industry development.

The comfort level, and arguably the success, of analysis of hardware focused initiatives is the similarity with previous industrial processes. In other words, there may be greater general similarities in the social, economic and policy implications between manufacturing steel and fabricating semiconductors than there is between producing hard disk drives and a software services industry. This means that analysis that purports to be of about high-technology development is generally just a subset of theories of industrialization, updated to focus on one of the most dynamic sectors of the global economy (as in Henderson 1989, Angel 1994, Dosi et al 1998). Within such analysis the failure to understand IT is not an issue, because the factors that have supported hardware strategies have not been about creating innovative regions. Such past failings are increasingly difficult for both governments and analysts to obscure as software and software practices become increasingly important within the global environment.

This means that focusing regional development initiatives from hardware to software may not be merely an industrial choice, but one of type of development, where success is about creating an environment where multiple types of new knowledge sectors can be developed, and traditional industries can be revitalized. A short thought experiment helps highlight this difference. Consider a regional strategy focused on developing a rice cooker for export in the current global environment. A traditional industrial focused strategy would focus on labor-cost, export markets, access to capital and access to raw materials in order to generate employment and export earnings. However, an informational strategy would focus on linking such development to a broader synergistic innovative environment. This would mean that the design of the rice cooker would involve the use of CAD and CAM tools, possible locally developed. Producing the highest value added products would entail incorporating IC and software features as well as perhaps embedding basic Internet connectivity. This would require knowledge of multiple markets and local standards. Even the basic production process of the rice cooker itself, to avoid pure labor-cost competition, would most likely entail Just-In-Time supply logistics and Total Quality Management (including work teams). These basic features of information management would entail the use of regional, and most likely, international telecommunications, especially the Internet. Even more possible is the use of the Internet as a central sales channel, perhaps even involving "build to order" processes or B2B management linked to global markets.

While stylized, this example helps us understand the overall patterns that restructure development strategies in an informational setting, even if focused on "traditional" industrialization. Software strategies differ most significantly in a focus on process as well as product and a concern with tacit as well as explicit knowledge. This shift in focus has a direct impact on governmental policy and the overall organizational structure of both firms and industries. It thus becomes essential to define and clarify the characteristics of successful IT strategies, as well as the specific nature of hardware and software strategies within them.

i Features of Innovative Regions: Developing a Matrix of Comparison

Theories of the informational economy and society are well outlined (Amin 1994, Webster 1995, Castells 1996a, Dicken 1998, Held et al 1999). Elaborating from these general theories, it is possible to develop a matrix of the basic features "innovative regions" or "technopoles" have that support successful development initiatives in general (Castells and Hall 1995, De Vol 1999 and 2000, Mytelka 1999). Such a matrix allows for an understanding of which factors need to be in place to support IT regional development, as well as the features that can be carried over from previous successful development strategies.

In general, IT focused development strategies should focus on three simultaneous goals: reindustrialization as linked to a constant transforming of comparative advantage, regional development to insure equitable and broad growth, and the building of synergy to create new economic opportunities and vitality (Castells and Hall 1995). In practice these goals evolve within the categories of public policy, comparative location benchmarking, and social infrastructure development. In terms of each of these, the state is the central catalyst. "State and local governments, public policies, and the interaction between private and public sectors are crucial for the genesis, expansion, and the fortification phases of high-tech development (De Vol 2000:3)." As such, governments should:

- Establish a clear and consistent development strategy and long-term vision, including internal and external promotion, and benchmark and evaluation procedures.
- Create public/private partnerships around this vision.
- Establish central policy and regulation to avoid competing spheres of influence.
- Foster the development of research, including universities, private labs and incubators and insure mechanisms for commercialization of this research.
- Foster the development of financial resources, especially venture funds and local angel networks.
- Foster the development of human resources, including entrepreneurial programs, life-long learning and migration.
- Capture national and international resources to fund and promote local objectives.
- Participate in national and international networks to promote local objectives.
- Develop synergistic policies focused on long-term aims and the sources of innovation, including linked industries (i.e. component and application firms), the creation of local and global niche markets, targeted FDI or branch-plant investments, and the production of unintended "spill-over" effects.

In terms of a simple matrix in which to evaluate the factors that support information regions and strategies, the above can be distilled to five factors whose "genesis, expansion and fortification" are central to the fostering of a high-technology region:

1. research centers,
2. investments (i.e. tax rebates) to jump-start development,
3. human-capital development,
4. venture capital for commercialization of research,
5. a culture and social environment that promotes development.

In terms of evaluating hardware and software strategies, the relative importance of specific factors of the matrix will clearly vary, or be absent altogether depending on regional history, resources and capacities. However, the absence or presence of specific factors that promote IT regions are exactly what needs to be analyzed in order to evaluate how such regions are constructed, transformed and sustained over time. Such a comparison allows for an understanding of the similarities or differences that structure hardware or software strategies specifically. It also helps contrast industrial and informational focused development as a generalized phenomenon, while holding the industrial sector constant.

ii High-Technology Hardware Focused Development Strategies

The extensive literature of successful state-guided high-technology hardware manufacturing (Deyo 1987, Henderson 1989, Borrus 1996, Naughton 1997, Ernst 1997, Lubeck and Eischen 1998, Rasiah and Best 2000) provides an excellent counterpoint to begin an analysis of software focused development initiatives. The extensive analysis and real world success of hardware focused strategies presents a kind of academic and policy path dependence that is essential to understand in order to adequately evaluate software strategies. Any state seeking to initiate or extend development under information technologies natural looks to the successful strategies of Taiwan, Korea, Singapore, China and Malaysia, amongst others, as models of successful development. The extensive analysis of these development initiatives has created a standard formulation of the role of the state in initiating and promoting development (World Bank 1993 and 1998).

On a first general pass, both hardware and software development initiatives do contain all of the elements that have come to define expanding global production: national economic liberalization focused on FDI and export promotion, global high-technology firms seeking low-cost and high-quality labor, regional tax and investment incentives, high-technology focused export processing zones, the spatial and organizational disaggregation of production processes globally, and a networking of high and low value-added regions within the global economy. Yet, hardware production networks have a number of differences that structure the role of the state and economic development quite distinctly from software. These differences are highlighted within the internal nature of hardware production, the supporting state and industrial environment and the geographic distribution of production.[12]

IT hardware sectors, no matter how technologically advanced, are fundamentally manufacturing focused. The internal processes that regulate the production of an integrated circuit is more complex, but not fundamentally different from the production of other industrial products like hard disk drives or steel or cars. Processes are rationalized, that is quantifiable and predictable, with a detailed and distinct understanding of each production phase from design to full-scale manufacturing (Sturgeon 1997b). This creates the potential for geographically or organizationally disaggregated production processes. Efficiency and costs are controlled through extensive rules and process management, including total quality control and just in time logistics. In this way, production deadlines and quotas can be efficiently met and controlled. There is a separation between management and worker, where the overall production process is obscured for labor. While workers may participate in problem solving or team related work, such activities are limited to that part of the production process under their control (Womack et al 1991). This creates a hierarchical organizational model of centralized control.[13] Traditional considerations of economies of scale, scarcity, supply management, long term return on investment, and barriers to entry structure the industry overall.

Because of this rationalized, hierarchical and traditional economic structure, state policy and the overall environment in which development takes shape are much closer to previous forms of industrialization (Amsden 1989, Johnson 1982, Gerschenkron 1962, Applebaum and Henderson 1992, Haggard 1990). This means that policies and expertise that guided the development of other industrial sectors can be more easily applied to high-technology hardware manufacturing. Tax incentives, large scale investments in plants and infrastructure, the production and control of semi-skilled industrial workforces, and management of macro-economic variables dominate state-policy in the promotion of hardware industries, whether local or foreign investment driven. Policy is structured around capturing

[12] It is important to note that these are general observations that clearly can be challenged by individual case examples. However, the goal is to contrast high-technology hardware production with software, not to understand how hardware itself is changing, which is outside the scope of this analysis. As has been noted, software is increasingly coming to structure all production processes in both hard and soft industries, and as such may fundamental blur, but not invalidate, the dichotomy set up here.

[13] This may contradict conventional analysis of both IT industries and the global economy (Held et al 1999, Dicken 1998, but it is entirely possible that decentralized external networks relationships may link organizations that are internally centralized. Additionally, many hardware products are produced through a combination of design and manufacturing, which may entail a combination of decentralized tacit processes (for research and design) and rational, defined, controlled practices (for manufacturing).

large scale manufacturing investments. In other words, states seek to permanently restructure the flow of global trade through specific and long-term hardware manufacturing investments. This focus produces an environment structured around employment generation, basic widespread educational skills, and local value added.

The combination of rationalized processes and state policies designed to capture production produces the distinct variable network geometry of hardware production (Borrus 1996, Dicken 1998, Held et al 1999). In other words, the rationalization of the hardware production process allows for a geographic separation of production into disaggregated production networks. Furthermore, because hardware processes are so well defined and elaborated, phases of production can be matched to specific regional resource matrices, including labor quality, state-policy and investment horizons. This means that regions do capture long term hardware investments, but these investments may be specifically targeted to the initial resource base of the region. Therefore depending on the firm and state policy, these investments may only reflect certain value added activities and limit the transference of complete or higher value added activities to the region over time.

This overall structure has shaped the globalization of high-technology hardware production. Outsourcing, both locally and globally, has been a key competitive strategy for firms in many sectors of hardware manufacturing (Sturgeon 1997a). International labor cost differentials combined with relatively high skill and quality levels have created a beneficial combination for many sectors. U.S. firms have remained competitive internationally by using these disaggregated production networks in order to move competition towards innovation and standards setting based on products rather than process (Borrus and Zysman 1997). The general trend has been for the separation of product and process innovation across regions within global production networks, leading to regional specialization around design or production. Much of this separation has been predicated on the clear rationalization of production — that is a systematic, disciplined, quantifiable understanding of production processes.

In terms of the basic matrix of innovative regions outlined above, there is a question of whether even the most successful hardware focused strategies outlined above have developed the infrastructure that would support innovative regions, whether hardware or software focused. First, local, independent research centers have not been a central concern of these hardware initiatives. Technology has either occurred within firm directed research labs or transferred through investments in equipment and network access. While there is some chance for knowledge to be transferred through local employment, the distinct rationalization of the production process has separated out product and process innovation. As product innovation has increasingly come to be a key center of value-added this limits the ability of local regions to capture innovative capacity. Furthermore, technology or innovative capacity that is transferred is general “locked-in” in terms of application only within the specific hardware industry that it comes from. This is especially true if the knowledge is separated in process or product categories. This means that creation of independent firms or research will be structured by the same barriers to entry, namely large scale investment, that structures hardware production in general.

Second, clearly successful investment strategies have been formulated and can be applied to non-hardware focused firms. However, the software sector is not a capital intensive industry. Physical investments are minimal, meaning that such strategies will be much more directed at profit incentives than investment support. Third, the investment that is needed involves not the firms directly, but the overall environment in which firms will locate. Central are investments in telecommunications, especially satellite uplinks and human resources. In regions with successful hardware strategies, the national telecommunication networks are generally quite good, with increasing emphasis on fiber optic and satellite networks. It is in terms of human capital that there are weaknesses. Hardware focused strategies have focused on establishing basic minimal educational standards for the population as a whole. Specialized higher education has tended to produce non-scientific or engineering talent, linked to administrative work. A software strategy, or innovative region in general, requires that the quality of skilled labor be at a minimal global level and at a quantity that will sustain investments for the near term. Such quality and quantity take time to develop, creating the paradox that investments of this type should actually take place years before they will be needed. In regions with successful hardware strategies, the immediate opportunities for students limits the effectiveness of such initiatives. In regions just developing, the opportunity cost of such a strategy may be prohibitive.

Fourth, the creation of local venture capital funds, especially private, has been extremely limited. The scale of investment needed, the time horizon required for hardware investment ROI, limits on the commercialization of local research, and tight government control of capital has limited the creation of knowledgeable and robust venture funds. Lastly, perhaps the strongest and weakest legacy of hardware focused strategies lies within the creation of an environment of innovative development. Clearly, the most successful regions have developed an excellent environment to manage and promote development. On the other hand, linked to the issues above, they have been far less successful in developing an ecology of innovation or an innovative milieu that can be self-sustaining. This brings into question the ability of regions to formulate an adequate strategy and vision, given past institutional and policy frameworks that are maximized to promote successful industrialization.

Operating on the assumption that software was structured equivalently to hardware industries, states and theorists have encouraged software focused development using the same set of policy tools and objectives. However, on deeper inspection the basic structure of hardware production and its geographic distribution raise questions about the ability of governments to institute software focused strategies along similar patterns. On a basic level, many of the key resources needed to develop innovative "information regions" are not present. In turn, the capacity of states to promote and manage software-focused development comes into question.

iii High-Technology Software Focused Development Strategies

As stated above, software strategies should seek to capture within regional centers the sources that drive innovation globally. In this way, software strategies distort and challenge previous understandings of national and regional development under globalization. Competitive advantage in software is created almost entirely through highly trained and innovative human capital. Extensive case studies have been conducted on the development of the software industry on national and regional level including Japan (Cusumano 1991), India (Heeks 1999), the US (Langlois & Mowery 1996), Bangalore (Parthasarathy 1999, Malecki and Oinas 1999), Ireland (O'Riain 1999), Hyderabad (Eischen 2000) and Singapore (Yeung 1999). Yet none of these studies has explicitly considered the unique production characteristics of software as detailed earlier. Importantly, these studies lack a clear analysis of the unique development processes, resources and geographic organization of the software industry that may profoundly shape and impact development possibilities (Hoch et al 2000, Brooks 1987, Raymond 1998). More importantly, no general pattern of software based development strategies, as separate and distinct from hardware, has been formulated. Significantly, this means that no comparative understanding exists of how national and regional software strategies have differed, and how these differences have structured the success or failure of these initiatives over time.

Internally, as detailed above, software development has a unique production process that requires large numbers of highly trained professionals working in close geographic proximity. Production of software, beyond the design and prototyping phase, is basically replication of one master copy. In this way, software development does not have a clear split between product and process innovation as in the case of hardware. Even where design and development are divided, the constant need for clarification between the two levels pushes for close physical proximity between software designers and developers. The process usually involves the creation of teams operating within the same location, or the actual combination of designer and developer within the same person. This means that once established, regional software sites capture the entire development process and the overall skills of software production.

Software also models rationalized, but increasingly complex social processes like financial transactions or the logistics of hardware production networks, that require extensive knowledge of the product domain itself. Much of this knowledge, and thus the knowledge of software development, is tacit and untransferable. Even within the general patterns that link together manufacturing broadly, software designed for manufacturing integrated circuits can not be easily transferred to the production of keyboards, and may not even be able to be transferred between two IC manufacturers. The lack of rationalization and the implicit nature of domain knowledge make it difficult to speak of software "manufacturing" or software "factories" (Cusumano 1991).

Organizationally, the blurring of design and production and the intense need for domain-knowledge makes the out-sourcing of lower value-added parts of the production process a very different proposition when compared with hardware production. Firms are most commonly organized along fully integrated product lines, keeping the full development process in-house, creating not only regional agglomerations of software development but firm domination of specific products within the software market. Where spatial disaggregation does occur, it is along product and not process lines. Firms, such as Microsoft or IBM, may have multiple sites of software development, but each will be responsible for the full development process of a specific product (i.e. operating systems, productivity software, e-commerce applications).

The overall impact of software investment then differs directly with hardware. Any region that develops a software industry or attracts a software firm essentially captures the entire production process. Meaning that designing even the most basic software product enables individuals to understand the process for developing software in general. The technological level of the product being developed does matter. If local software developers can participate in leading-edge innovation, they will have captured the full capacity to replicate this process on their own. However, the need for domain knowledge gives local programmers an advantage in creating products that may not be the leading but bleeding edge, that is where technical skill and domain knowledge blend together to develop innovative products.

Given the nature of software development, finding and retaining software professionals is the essential constraint facing software firms. As previously stated, the demand for software services has consistently outstripped the supply of labor or any development process improvements. In response to the limits on software rationalization, software development and the industry as a whole have come to focus on an extensive growth strategy, where new projects and demand have been met by a constantly expanding labor pool of skilled software professionals (Cusumano and Selby 1995, Hoch et al 2000, Karolak 1998). This consistent need for extensive growth, for a constant input of large amounts of skilled labor, has been one of the central aspects shaping the regional and global development patterns of the industry overtime. As such, software globalization, in contrast to hardware, has been structured by the search for factors of production, rather than their lower cost. This has created an industry that thinks in terms of a global labor market, and views regional investments as essential aspects of tapping into human resources and local domain knowledge.

Software firms have been inserted into and created broader national and global networks of labor and immigration consistently over the last twenty years. Innovation and information have been managed through the development of social networks that have linked vertically integrated, geographically specific firms with new regional sources of software professionals (Saxenian 1999) on a global basis. Through a constant expansion to include new individuals and regions, these social networks have provided software firms the strategic benefits of networks within the limits of software production (Bhaskar 2000). The limits of this labor-based globalization combined with the need for domain knowledge creates spaces in which states, on regional or national levels, can capture new investments as software firms invest globally in regional software centers. Firms have retraced the paths of these global labor flows, locating investments in regions that have traditional been sources of high-technology talent. They have also tapped into the social capital of their employees in making location decisions and successful launching new development sites.

For states seeking to develop a software focused strategy, the understanding of software processes and globalization create an entirely new framework for policy than previous hardware focused strategies (Heeks 1999, Eischen 2000). In terms of the matrix of innovative regions above, each of the five factors is essential to creating an environment that will support a software focused development strategy. The nature of software, especially the innovation around codifying new domain knowledge, requires that regions develop these factors in order to sustain a software focused development strategy. First, regions with already established research centers are more likely to develop successful initiatives. These research centers do not necessarily have to be IT focused, but, in the case of military research or biotechnology or finance, can contribute to the production of skilled labor and an environment of innovation. They are also crucial to providing the domain knowledge and mechanisms for application of technologies to local needs even if conducted within foreign firms.

Second, basic investments in quality of life measures, telecommunications and strategic partnerships can be undertaken. If these already exist, then incentives can be extended to firms in the form of tax breaks. Traditional considerations of economies of scale, scarcity, supply management, long term return on investment horizons, and barriers to entry either do not exist or function in the same way for software firms. Long term tax breaks or holidays are not linked to large-scale capital investments, but limited to profits and rents. Most likely, such policies will entail a transfer of funds to investing firms, rather than a lessening of revenue to the state. However, software firms, and firms operating within software process environments, are far more concerned with access to quality labor.

As such, the third factor of human-capital development is crucial. Building large-scale technology parks with high-speed data links will not attract investment if a large pool of software professionals is not readily available. As stated above, both quality and quantity are essential. Quality can be improved through international partnerships with leading universities and firms to create joint degree or certification programs. This can also be extended to joint-research efforts. The quantity of labor is much more difficult. Regions with a legacy of producing large quantities of underutilized engineers will have a distinct advantage. This opens up the possibility for regions with distorted educational systems in relatively undeveloped regions to aggressively market themselves as sites of software development.

Fourth, the unique advantage posed by the stages of software globalization, first through global labor networks and second as locating in the home regions of the labor pools, opens up possibilities of developing local venture capital networks. Many émigrés will lead their firm's investments, bringing both knowledge and needed capital. Others will want to develop ties back home that will focus on new angel investments. These networks create natural mechanisms for developing local angel and VC networks especially within regions with strong social ties.

Lastly, regions with strong social capital are much better positioned to be successful in building a regional development strategy. The question of creating a culture of innovation, as above in the case of hardware, is linked in many ways to the first four factors. However, regions with local culture of risk taking, competition and entrepreneurship will be able to mobilize resources to embed the development process with local dynamism (Lubeck and Eischen 1998).

Overall software presents unique opportunities to regions in contrast to hardware production. Economies of scale and scarcity do not apply in an equivalent fashion in software production, meaning that no clearly defined minimum efficient scale or that specific fixed resources, outside of skilled-labor, are required for production. Production, once established, does transfer the entire production process to the region. From the simplest HTML document to the most complicated financial product, the software development process is essentially the same. The low barriers to entry in the industry, outside of marketing and sales, allow for the easy development of local firms. This, however, is contingent on the ability of a region to retain skilled labor and develop entrepreneurs. Investments by states in higher education are difficult to recoup if individuals are able to migrate to higher wage and quality of life regions, working for the same firms. This means that states must create environments where entrepreneurs want to live, and invest in infrastructure that will enable them to market and sell their products globally.

Long-term competitiveness in software is centered on the creation of dynamic comparative advantage. Any successful strategy must focus on the development of new sources of labor and innovation to constantly adjust to new opportunities and challenges in the global market. In this way, software is about creating an innovative milieu that is institutionally supported by new research, high-skilled labor and mechanisms for commercializing these resources. Software development involves the specification and mathematical mapping of unique processes. This means that software must continually be adapted to new environments, be they geographic, social, cultural or organizational. This process of adaptation means that software development processes will continue to expand to regions around the globe as IT is brought to focus on an ever more diverse and complex problems. This creates a situation of both opportunity and challenge for state-led development initiatives. On one hand it insures the opportunity for the creation of local software industries, if only to capture local domain knowledge to both translate existing software and respond to local needs. This process is reinforced by the agglomeration and skill transfer characteristic of software development. On the other, it creates a series of challenges based on the unique organizational structure and development process of software. These will create new regulatory and social challenges for governments quite distinct from even the most successful hardware focused strategies.

The challenge is creating the initial environment in which such global processes can take root, and then sustaining such processes over time. Initiatives centered on such projects have been and are extremely difficult to develop (Castells and Hall 1995). Drawing on the existing cases and analysis (Malecki and Oinas 1999, Barnes and Gertler 1999, Storper and Harrision 1990, Scott 1996, De Vol 2000, Castells and Hall 1994), a preliminary summary of aspects of hardware and software regions is given in the following table.

Contrasting Aspects of Hardware and Software Focused IT Environments

Aspect	Hardware	Software
Research	Basic, semi-essential	Problem specific, probably market focused, essential
Investment	State, Banks, Bond Markets, FDI	Angel, VC, Capital markets, Alliance, FDI
Institutions for Application of Technology	National Research Institutes, Military	VC's, Incubators, Start-up firms
Quality of Life	Highly important, but focused on general social standards	Highly important, but selectively focused on digerati and individuals
Business Culture	Bureaucratic, Rule-based, Developmentalist	Innovative, Risk oriented, Profit focused, Applied
Training	General and fixed in duration	Specific and continuous on new technologies
Telecommunications	Basic phone service, State owned media, limited cable	Fiber optic/cable service, privatized media, satellite uplinks
Education	General literacy, some elite training for administrative and management purposes	Specialized and elite training in IT fields, university and above focused
Role of State	Developmentalist	Facilitator/Protector
Foreign Investment	EOI focused with FDI essential	Research focused with both FDI and alliances key
Capital Flows and Markets	Restricted and national	Open and linked
Information Flows	Restricted and controlled	Open and linked
Industry Structure	Hub and Spoke, Monopolistic (via resources), Hierarchical	Networked, Monopolistic (via standards or market share), Flat
Firm Structure	Bureaucratic, Hierarchical, Mass production focused	Flexible, Networked, Horizontal, Craft-knowledge forming focused
Constraints on Technology Development	Patents, physical limitations	Creativity, Social and regional linkages, Human movement
Constraints on Technology Application	Lack of investment capital, Slow adoption, Different comparative advantage, Low FDI	Lack of commercialization channels, Lack of venture capital
Immigration	Brain drain	Knowledge transfer, Network building
Role of Culture	National emphasized	Essential as product
Role of Regions	Downplayed to national aims	Essential for localized implementation and tailoring of policies

iv ***Building a Software Region: Opportunities and Limitations***

Creating a software focused development strategy means moving toward an understanding of the basic processes, products and industry characteristics of software. It is also a move to framing development questions in terms of the basic organizational, social and development logics accompanying information centered economic growth. The overall strategy must understand that software entails domain knowledge (the in-depth understanding of field specific processes and outcomes) embodied in skilled individuals within an environment that can translate this knowledge into actual marketable products. This means that the strategy must insure the attraction of people and investment that understand both the organizational and technical aspects of developing such products. And perhaps most importantly, it means developing a self-sustaining pattern of growth based on the creation of new knowledge and people through research and education. Simply, the strategy must move to create an ecology of innovation. Such strategies may directly contradict previous hardware strategies, common sense or short term goals. They may also have serious social consequences that contradict previous development strategies and aims. The ability of governments to generate policies that evenly distribute the gains from software focused strategies, while maintaining the organizational and social structures that provide those gains, will be one of the greatest challenges of new regional initiatives.

What this means in terms of the overall social structure and economic development of a region, remain to been seen. The overall social transformations linked to the general process of industrialization – the rise of a middle-class, urbanization, consumerism, increased gender equity, expanded literacy, social movements – that have accompanied high-technology hardware industrialization (World Bank 1993, Amsden 1989, Lubeck and Eischen 1998) has yet to be established as the default pattern for software focused strategies. Overall then, different forms of investment and policy, both in building and maintaining a software industry, are required from states that focus on developing a software industry. In contrast to previous hardware centered strategies, there is no clear trend or method that has demonstrated the capacities and policies states should develop to manage these opportunities and challenges (Digital Economy 2000, Lessig 1999). The formation of new software development centers globally has replicated the labor and firm agglomeration patterns that have characterized the industry's development in places like Silicon Valley (Parthasarathy 1999). The most established of these regional centers of software development – chiefly in Ireland, Israel and India – have come to replicate the social and economic tensions present in more advanced IT economies and regions (Castells 1994, Held et al 1999, Dicken 1998),[14] starting from a significantly lower level of development and with much lower levels mechanisms for social protection (Eischen 2000).

a Potential Consequences of Software-Focused Development

One of the central ironies of a software centered development strategy is that policies and institutions that are weaknesses under national or industrial conceptions of development – chiefly skewed educational systems, poor infrastructure and regional factionalism – can become strengths under a software based development model. Regions can build on national development legacies while targeting specific investments in infrastructure, telecommunications and education that institutionalize and expand on some of the legacy's most distortive aspects. Under these conditions, there is serious potential for a permanent digital divide (where access to technology, education and incomes are highly correlated), not between nations or regions, but between social classes, cities or even neighborhoods, as inequalities are institutionalized under a software focused strategy.

Strong and socially consensual policies are needed to embed such developments in the broader society rather than merely set up a parallel social and economic reality. If such policies do not exist, the risk is that the general spread of telecommunications, electricity and educational services will remain relatively stagnant, while new institutions and targeted investments link small sections of the society to the global economy. While many of these problems exist for hardware focused strategies as well, the traditional benefits of industrialization and FDI, specifically employment and export promotion, are fundamentally different for software. This challenges states to develop innovative policies, with very few successful models to follow. Even the most successful information technology regions, like Silicon Valley, face completely new social tensions and inequalities (Benner 2000).

These are not temporary problems, but general patterns that will continue to occur within the global economy. Software is increasingly an essential feature of the global economy. Governments will, and should, focus on software as an essential sector to incubate locally (Parthasarathy 1999), and new centers of software development will arise from relative obscurity. Regions can initiate software based development and address social inequalities simultaneously by focusing on the long-term innovative capacity of the region. However, long term innovative capacity in software is not captured in physical

[14] It is important to note that social and economic patterns surrounding software regions are appear to exist in other regions as well, including Los Angeles (Film), San Francisco (Biotechnology and E-Commerce), Austin (Computers and ICs), New York (Finance), Bombay (Film), Hong Kong (Finance, Ecommerce, Film).

resources, but in human capital that must be constantly recreated and revitalized through education in an industry with extremely short innovation cycles. The opportunity around software, and industries following software production patterns, is that local knowledge is needed. And once begun, the nature of software production will most likely produce agglomerations of skilled professionals able to capture the full production process. These two factors open the possibility for applying global skills to local needs addressing both innovation and social inequality simultaneously.

The key aspect is that regions should recognize that they are really developing a mastery of information or innovative processes, and not software itself. In other words, focusing on synergy and developing the basic structures that promote an "ecology of innovation" will produce economic and social development far outside of just software. Software is and will remain increasingly at the center of IT. However, it is the blending of software with other industries and domain knowledges that produces the real dynamism in the software industry. Understanding this opens up the real possibilities for regional development strategies. Existing industries and research centers, from media and film to agriculture and education, can become an integral part of a successful strategy, expanding the impact of the development process and insuring its sustainability over time. All of this combined will establish the embedding of local knowledge into broader national and global processes, while insuring that the most negative social and economic divisions are ameliorated.

VI In Conclusion: Developing Models, Methods and Research Programs to Understand IT and Development

The three quotes that began this analysis offer entry points into considering IT and its impact on issues of social and economic development. Ridenour describes the process of how computers are capable of being, and as detailed above, have become "information machines". The key technological development of the last thirty years is exactly the creation of information technology as a general phenomenon with ever greater distribution and connectivity. The key aspect is not, as is usually focused upon, technology, but rather information. This means that the increasingly connected and digitalized world we inhabit is structured by the flow of information, and how that information, as socially determined, is congealed into new forms of profit and power.

Von Neuman, speaking over fifty years ago of science and technology, helps to explain the basic patterns of economic and social development as issues of structure, organization, information and control under these informational flows. An information economy restructures social relationships in order to maximize the generation, manipulation, dissemination and commercialization of information. This results in new organizational and institutional imperatives, and as highlighted by the demands of creating an IT industry, that offers both new opportunities and challenges to regions. A distinct pattern of production, as outlined in the case of software development processes, models the way in which knowledge and information are produced and valorized into power and profits in the global environment.

The final quote describes this entire process, in essence what the information economy is, in three short sentences. An Irish subsidiary of IBM has developed a web based real-time translation program based on Java. The knowledge to create such a program resides in the programmers who created it, that is in Ireland, even though they work for a global firm. The product is entirely software based and only functions on the Internet, where it is instantly available globally. The program enables real time communication, regardless of location or language. What these languages are, how well they are translated and how they structure communication is a function of the domain knowledge and assumptions of the programmers as structured in the program code. The ability to regulate the product is inherent in the coding of the program and priorities of IBM, and not any government or social process.

This field has attempted to define and clarify IT in order to understand how simple stories, like the case of IBM, hide very complex issues that are at the heart of multiple trends in the global environment. Such clarifications and definitions are essential, because much of the analysis currently available either fails to capture the complexity of these simple stories or to link them to broader theorizations of globalization or the "information society". The main critique of existing theories is that they fail to understand either the technological or informational aspects of IT, either historically or in practice. This leads, for an example, to a complete lack of understanding of the importance of software as a significant industry within IT on a simple level, and as perhaps the central way to define and understand IT on a higher level. Many of these failing stem from not approaching software or IT sectors through a methodology that simultaneously considers process, product and industry dimensions. The ambitious aspect of this field is that it has tried to both critique existing theories, and offer analysis that avoids the same failings.

At the very least, software is an essential sector within the IT industry. Because of the demand of its unique production process, the software industry will continue to expand globally, and in the process create new regional software centers. How states and firms respond to this opportunity will be an essential issue for economic and social development in the coming decades. On a higher level of abstraction, the increasing presence of software in industries and institutions outside of IT requires a careful consideration of the software practices and structures that increasingly impact upon social and economic interactions generally. In this sense, understanding software may be a basis for clarifying the broader production processes, organizations and institutions that help structure new global trends. This opens the possibility for considering software as the ideal type of process, product and industry within the global environment. As such, a detailed analysis of software may be a way to ground theorizations of globalization and the "information society" in concrete practices and micro-level events.

Such a stepwise approach has already been conducted for multiple industries within the global environment, specifically hardware industries like semiconductors. A future research agenda should focus on creating an equivalent analysis for software in order to test hypotheses and theories of the "information society" in an empirically rich and concrete manner. It would also include recasting these earlier studies in terms of a basic methodology of process, product and industry, in order to define exactly what is the informational aspect of these sectors. This would, just as in the case of software, help ground theoretical considerations in detailed empirical case studies. Through such an approach, the general patterns that govern global economic and social trends could be uncovered. As such, this would allow analysis of IT industries and processes to be informed by similar non-technical industries in fashion, film, media and biotechnology to name just a few. Such comparisons would help elaborate on basic theorizations of innovation, information, organizational forms, management and development. Perhaps most importantly, it would provide a direct link into policy and analysis through a consideration of basic processes and skills that overlap divergent industries, and thus would help elaborate on how broader economic and social development might be encouraged. This would be especially true in the case of developing economies, where IT may or may not be an appropriate development focus.

The central failure of social science, particularly sociology, has been the failure to recognize the central role of domain knowledge within information technology. The informational aspect of IT does not signal only communication, but the embodying of social practices and interactions in technology itself. Information technology is the process of converting social knowledges and practices into digital form, so that it can be manipulated, disseminated and controlled within a coded binary architecture. This process of translation is really a process of transformation. Norms and values are implicitly included in the algorithms that shape the coding of the social processes represented by particular domain knowledges. In this way, IT increasingly shapes the rules and procedures under which social processes take place. Code is thus a new form of law, one that is flexible, but also inherently social.

Kranzberg states that "technology is neither good, nor bad nor neutral (1985)", meaning that society determines the application and impact of technology in all cases. However, the case of IT, and software in particularly, is different. Software is the way in which knowledge or information is given form in a digital world, as such it embodies social knowledge. This means that prior to its application by society, software is already structuring the forms that social interactions can take. Combined with Metcalfe's Law[15], this implies that once software patterns of interaction become dominant– such as the protocols that shape Internet communication or define how computers structure information – the social interpretation or regulation of these technologies is already limited.

This reality underlies many of the fundamental social and economic debates occurring in multiple regions and nations. On the surface debates on privacy, biotechnology or online music seem like simple debates around intellectual property or other rules of law, but they are really debates about the fundamental architecture of a coded, binary, information society. Distributed computing models like Napster open up the possibility for a purely global, decentralized, peer to peer system of communication. Open source models of software development push for code, the laws that structure the IT architecture, to be made open and accountable to social scrutiny prior to their embeddedness in a technical architecture. Understanding IT in this way is crucial to developing accurate theories, analysis and policies of the current social and economic environment. And because the environment is global, a general understanding of IT goes very far in helping to explain the institutional and organizational trends, tensions, flows and linkages that structure economic and social interactions on multiple scales.

[15] Metcalfe's Law states that the "value" or "power" of a network increases in proportion to roughly the square (for large numbers) of the number of nodes on the network, or more precisely $N^2 - N$.

VII Bibliography

Agre, Phillip (1995a) “Institutional Circuitry: Thinking about the Forms and Uses of Information”, Information Technology and Librarians, Volume 14, Number 4, Page 225

Agre, Phillip (1995b) “From High Tech to Human Tech: Empowerment, Measurement, and Social Studies of Computing”, Computer Supported Cooperative Work, Volume 3, Number 2, Page 167

Agre, Phillip and Douglas Schuler (1997) Reinventing Technology, Rediscovering Community: Critical Explorations of Computing as a Social Practice, Ablex, Palo Alto

Amin, Ash, Editor (1994) Post-Fordism: A Reader, Blackwell, Cambridge, USA

Amin, Ash and Jerzy Hausner (1997) “Interactive Governance and Social Complexity” in Ash Amin and Jerzy Hausner, Editors, Beyond Market and Hierarchy: Interactive Governance and Social Complexity, Edward Elgar, Lyme, US

Amsden, Alice (1989) Asia’s Next Giant: South Korea and Late Industrialization, Oxford University Press, New York

Angel, David P. (1994) Restructuring for Innovation: the Remaking of the U.S. Semiconductor Industry, Guilford, New York

Appelbaum, Richard P. and Jeffrey Henderson, Editors (1992) States and Development in the Asian Pacific Rim, Newbury Park, Calif.: Sage Publications

Armour, Phillip G. (2000) “The Case for a New Business Model”, Communications of the ACM, August, Volume 43, Number 8

Barnes, Trevor J. and Meric S. Gertler (1999) The New Industrial Geography: Regions, Regulation and Institutions, Routledge, New York

Benner, Chris (2000) Navigating Flexibilities: Labor Markets and Intermediaries in Silicon Valley, Doctoral Dissertation, Department of Urban Studies and Regional Planning, University of California, Berkeley

Berlinski, David (2000) The Advent of the Algorithm: The Idea that Rules the World, Harcourt, Inc., New York

Bhaskar, T.L.S (2000) “The Telugu Diaspora in the United States”, CGIRS Working Paper, WP#2000-1, Center for Global International & Regional Studies, University of California, Santa Cruz, http://www2.ucsc.edu/cgirs

Boehm, Barry W. (1988) “A Spiral Model of Software Development and Enhancement”, Computer, May

Borrus, Michael (1993) The Regional Architecture of Global Electronics: Trajectories, Linkages and Access to Technology, Berkeley, CA: Berkeley Roundtable on the International Economy, University of California, Berkeley

Borrus, Michael (1996) Left for Dead: Asian Production Networks and the Revival of US Electronics, MIT Japan Program, Center for International Studies, Massachusetts Institute of Technology

Borrus, Michael and John Zysman (1997a) Globalization with borders: the rise of Wintelism as the future of industrial, Berkeley, CA: Berkeley Roundtable on the International Economy, University of California, Berkeley

Braun, Ernest and Stuart Macdonald (1982) Revolution in Miniature: the History and Impact of Semiconductor Electronics Re-explored, Cambridge University Press, New York

Britcher, Robert N. (1999) The Limits of Software: People, Projects and Perspectives, Addison-Wesley, Menlo Park, California

Brooks, Frederick P. (1974) “The Mythical Man-Month”, Datamation, December

Brooks, Frederick P. (1987) “No Silver Bullet: Essence and Accidents of Software Engineering”, Computer Magazine, April

Brooks, Frederick P. (1995) The Mythical Man-month: Essays on Software Engineering, Addison-Wesley, Reading, Mass.

Buchanan, R. A. (1992) The Power of the Machine: the Impact of Technology from 1700 to the Present, Viking, New York

Business Software Alliance (1999) Forecasting a Robust Future: An Economic Study of the U.S. Software Industry, June, http://www.bsa.org/

Cairncross, Frances (1997) The Death of Distance: How the Communications Revolution will Change Our Lives, Harvard Business School Press, Boston, Mass.

Cameron, Gavin (1998) "Economic Growth in the Information Age: From Physical Capital to Weightless Economy", Journal of International Affairs, Volume 51, Number Two

Campbell-Kelly, Martin and William Aspray (1996) Computer: A History of the Information Machine, Basic Books, New York

Carnoy, Martin, Manuel Castells and Chris Benner (1997) "Labour Markets and Employment Practices in the Age of Flexibility: A Case Study of Silicon Valley", International Labour Review, volume 136, number 1

Castells, Manuel (1989) The Informational City: Information Technology, Economic Restructuring, and the Urban-regional Process, Blackwell, Oxford

Castells, Manuel (1996a) The Information Age: The Rise of the Network Society, Cambridge, Mass.: Blackwell Publishers

Castells, Manuel and Yuko Aoyama (1994) "Paths Towards the Informational Society: Employment Structure in G-7 Countries, 1920-1990", International Labour Review, volume 133, number 1

Castells, Manuel and Peter Hall (1994) Technopoles of the World: The Making of 21st Century Industrial Complexes, Routledge, New York

Cohen, Stephen S., J.Bradford DeLong and John Zysman (2000), "Tools for Thought: What is New and Important About the New Economy?", BRIE Working Paper 138, E-Conomy Project, UC Berkeley, http:www.e-conomy.berkeley.edu/

Coyle, Diane (1998) The Weightless World: Strategies for Managing the Digital Economy, MIT Press, Cambridge, Massachusetts

Cusumano, Michael A. (1991) Japan's Software Factories: A Challenge to U.S. Management, Oxford University Press, New York

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

Davis, Alan M. (1997) "Software Life Cycle Models", in Software Engineering Project Management, Edited by Richard Thayer, IEEE Computer Society, Los Alamitos, California

De Vol, Ross C. (1999) America's High-Tech Economy: Growth, Development, and Risks for Metropolitan Areas, Milken Institute, www.milken-inst.org

De Vol, Ross C. (2000) Blueprint for a High-Tech Cluster: The Case of the Microsystems Industry in the Southwest, Milken Institute, www.milken-inst.org

Deyo, Frederic C., Editor (1987) The Political Economy of the New Asian Industrialism, Ithaca, N.Y.: Cornell University Press

Dicken, Peter (1998) Global Shift: Transforming the World Economy, London: Paul Chapman

Digital Economy 2000, Economics and Statistics Administration, U.S. Department of Commerce, Washington D.C.

Doray, Bernard (1988) From Taylorism to Fordism: a Rational Madness, Free Association Books, London

Dosi, Giovanni (1984) Technical Change and Industrial Transformation, New York: St. Martin's Press

Dosi, Giovanni, David J. Teece, and Josef Chytry, Editors (1998) Technology, Organization, and Competitiveness: Perspectives on Industrial and Corporate Change, New York : Oxford University Press

Downey, John and Jim McGuigan, Editors (1999) Technocities, SAGE, Thousand Oaks, Calif.

Drucker, Peter Ferdinand (1993) Post-Capitalist Society, New York, HarperBusiness

Edwards, Paul N. (1996) The Closed World: Computers and the Politics of Discourse in Cold War America, MIT Press, Cambridge, Mass.

Egan, Edmund A. (1996) The Era of Microsoft? Technological Innovation, Network Externalities, and the Seattle Factor in the US Software Industry, Berkeley Roundtable on the International Economy, Working Paper 87, University of California, Berkeley, http://brie.berkeley.edu

Eischen, Kyle (2000) Regional Development under Software Globalization: Software Development Processes, Human Networks and Social Capital in Hyderabad, Andhra Pradesh
Working Paper, Forthcoming at the Center for Global, International and Regional Studies, www2.ucsc.edu/cgirs

Ernst, Dieter and David O'Connor (1992) Competing in the Electronics Industry : the Experience of Newly Industrialising Economies, Paris, France : Development Centre of the Organisation for Economic Co-operation and Development ; Washington, D.C. : OECD Publications and Information Centre

Ernst, Dieter (1997) From Partial to Systemic Globalization: International Production Networks in the Electronics Industry, Berkeley, CA :Berkeley Roundtable on the International Economy, University of California, Berkeley : San Diego : Graduate School of International Relations and Pacific Studies, University of California at San Diego, Data Storage Industry Globalization Project report, 97-02. http://brie.berkeley.edu/BRIE/pubs/wp/index.html

Evans, Peter B. (1995) Embedded Autonomy: States and Industrial Transformation, Princeton, NJ: Princeton University Press

Fenton, Norman, Shari Lawrence Pfleeger and Robert Glass (1994) "Science and Substance: A Challenge to Software Engineers", IEEE Software, Volume 11, Number 4, July

Ferguson, Pat, Watts Humphrey, Soheil Khajenoori, Susan Macke and Annette Matvya (1997) "Results of Applying the Personal Software Process", Computer, May

Flamm, Kenneth (1988) Creating the Computer: Government, Industry, and High-technology, The Brookings Institution, Washington D.C.

Forester, Tom (1987) High-Tech Society: The Story of the Information Technology Revolution, MIT Press, Cambridge, Massachusetts

Gerschenkron, Alexander (1962) Economic Backwardness in Historical Perspective: A Book of Essays, Harvard University Press, Cambridge, MA

Gibbs, Wayt (1994) "Software's Chronic Crisis", Scientific American, Volume 271, Number 3, September

Gibilisco, Stan, Editor (1997) The Illustrated Dictionary of Electronics, McGraw-Hill, New York

Gilb, Tom and Roger Pressman (1996) Level 6: Why We Can't Get There from Here, IEEE Software v13, n1, January: 97

Gilb, Tom (1996) What is Level 6, http://stsc.hill.af.mil/swtesting/gilb.asp

Gilb, Tom (1997) Evolutionary Project Management Handbook, http://home.c2i.net/result-planning/

Giordano, Lorraine (1992) Beyond Taylorism : computerization and the new industrial relations, St. Martin's Press, New York, NY

Gordon, Richard (1994a) "Strategic Alliances and Structural Change in Silicon Valley's Semiconductor Industry", Working Paper 94-3, Center for the Study of Global Transformations, University of California, Santa Cruz

Gordon, Richard (1994b) "State, Milieu, Network: Systems of Innovation in Silicon Valley", Working Paper 94-4, Center for the Study of Global Transformations, University of California, Santa Cruz

Grady, Robert B. (1994) "Successfully Applying Software Metrics", Computer, Volume 27, Number 9, September

Guice, Jon (1997) Designing the Future: the US Advanced Research Projects Agency and the Revolution in Computing, Dissertation, UC San Diego,

Hafner, Katie and Matthew Lyon (1996) Where Wizards Stay Up Late: the Origins of the Internet, Simon & Schuster, New York

Haggard, Steph (1990) Pathways from the Periphery: The Politics of Growth in the Newly Industrializing Countries, Cornell University Press, Ithaca, New York

Haraway, Donna Jeanne (1997) Modest-Witness@Second-Millennium.FemaleMan-Meets-OncoMouse: Feminism and Technoscience, Routledge, New York

Harvey, David (1990) The Condition of PostModernity, Blackwell, Cambridge MA

Heeks, Richard (1996) India's Software Industry: State Policy, Liberalization and Industrial Development, Institute for Development Policy and Management, University of Manchester, Working Paper #3, October, Sage Publications, Thousand Oaks, CA.

Heeks, Richard (1999) Software Strategies in Developing Countries, Institute for Development Policy and Management, University of Manchester, Working Paper #6, June, http://www.man.ac.uk/idpm

Held, David, Anthony McGrew, David Goldblatt and Jonathan Perraton (1999) Global Transformations: Politics, Economics and Culture, Stanford University Press, Stanford, California

Henderson, Jeffrey William (1989) The Globalisation of High-Technology Production: Society, Space, and Semiconductors in the Restructuring of the Modern World, Routledge, New York

Herbsleb, James, David Zubrow, Dennis Goldenson, Will Hayes and others (1997) “Software Quality and the Capability Maturity Model”, Communications of the ACM, Volume 40, Number 6, June

Hoch, Detlev J, Cyriac R. Roeding, Gert Purkert and Sandro K. Lindner (2000) Secrets of Software Success, Harvard Business School Press, Boston

Humphrey, Watts S. (1987) Managing for Innovation: Leading Technical People, Prentice-Hall, Englewood Cliffs, N.J

Humphrey, Watts S. (1995) A Discipline for Software Engineering, Addison-Wesley, Reading, Mass.

Humphrey, Watts S. (1997) Managing Technical People: Innovation, Teamwork, and the Software Process, Addison-Wesley, Reading, Mass.

Jalote, Pankaj (1997) An Integrated Approach to Software Engineering, Springer, New York

Johnstone, Bob (1999) We Were Burning: Japanese Entrepreneurs and the Forging of the Electronic Age, Basic Books, New York

Johnson, Chalmers A. (1982) MITI and the Japanese Miracle: the Growth of Industrial Policy, 1925-1975, Stanford, Calif.: Stanford University Press, 1982.

Jurgens, Ulrich, Thomas Malsch and Knuth Dohse (1993) Breaking from Taylorism: Changing Forms of Work in the Automobile Industry, Cambridge University Press, London

Kalakota, Ravi and Andrew B. Whinston (1997) Electronic Commerce: a Manager's Guide, Addison-Wesley, Reading, Mass.

Kalakota, Ravi and Andrew B.Whinston, Editors (1996) Frontiers of Electronic Commerce, Addison-Wesley, Reading, Mass.

Kalakota, Ravi and Andrew B.Whinston, Editors (1997) Readings in Electronic Commerce, Addison-Wesley, Reading, Mass.

Kambil, Ajit (1997) "Doing Business in the Wired World," Computer, May, IEEE

Karolak, Dale Walter (1998) Global Software Development, IEEE Computer Society, Los Alamitos, California

Kelly, Kevin (1998) New Rules for the New Economy: 10 Radical Strategies for a Connected World, Viking, New York

Kohanski, Daniel (1998) The Philosophical Programmer: Reflections on the Moth in the Machine, St. Martin's Press, New York

Kranzberg, Melvin (1985) "The Information Age: Evolution or Revolution?", in Information Technologies and Social Transformation, Bruce R. Guile, Editor, National Academy Press, Washington D.C.

Kurzweil, Ray (1999) The Age of Spiritual Machines: When Computers Exceed Human Intelligence, Viking, New York

Lash, Scott and John Urry (1994) Economies of Signs and Space, Sage, Thousand Oaks

Langlois, Richard N. and David C. Mowery (1996) "The Federal Government Role in the Development of the U.S. Software Industry", in The International Computer Software Industry: A Comparative Study of Industry Evolution and Structure, Edited by David C. Mowery, Oxford University Press, New York

Leebaert, Derek (1995) The Future of Software, MIT Press, Cambridge, Mass.

Lefebvre, Louis A. and Elisabeth Lefebvre, Editors (1995) Management of Technology and Regional Development in a Global Environment: an International Perspective, P. Chapman, London

Leonard, Andrew (2000) "Sneaky Algorithms", Webreview, http://webreview.com/96/03/22/edge/index.html

Legg, Deanna B. (1997) "Synopsis of COCOMO", in Software Engineering Project Management, Edited by Richard Thayer, IEEE Computer Society, Los Alamitos, California

Lessig, Lawrence (1999) Code: and Other Laws of Cyberspace, Basic Books, New York

Liebowitz, S. J. and Stephen E. Margolis (1999) Winners, Losers & Microsoft: Competition and Antitrust in High Technology, Independent Institute, Oakland, Calif.

Loader, Brian D., Editor (1997) The Governance of Cyberspace, Routledge, New York

Lubeck, Paul and Kyle Eischen (1998) "Silicon Islands and Silicon "Valles": Rethinking Mexican Regional Development Strategies in an Era of Globalization", Forthcoming in Las Nuevas Fronteras del Siglo XXI: Dimensiones Culturales, Políticas y Socioeconómicas de las Relaciones México-Estados Unidos, Edited by Alvarez, Castillo, Klahn and Manchón, UNAM/UAM

Malecki, Edward J. (1997) Technology and Economic Development: the Dynamics of Local, Regional, and National Competitiveness, Longman, Essex, England

Malecki, Edward and Paivi Oinas, Editors (1999) Making Connections: Technological Learning and Regional Economic Change, Ashgate, Brookfield USA

Meares, Carol Ann and John F. Sargent, Jr. (1999) The Digital Workforce: Building Infotech Skills at the Speed of Innovation, U.S. Department of Commerce, Office of Technology Policy

Miller, Riel (1996) Territorial Development and Human Capital in the Knowledge Economy: Towards a Policy Framework, OECD Working Papers, Volume 4, Number 64, Lead Notebook Number 23, Paris

Miller, Riel (1997) The Internet in Twenty Years: Cyberspace, the Next Frontier?, OECD International Futures Programme, Number 14, May, Paris

Mitchell, William J. (1998) City of Bits: Space, Place and the Infobahn, MIT Press, Cambridge

Moschovitis, Christos, Hilary Poole, Tami Schuyler and Theresa M. Senft (1999) History of the Internet: a Chronology, 1843 to the Present, ABC-CLIO, Santa Barbara, Calif.

Mowery, David C. (1996) The International Computer Software Industry: a Comparative Study of Industry Evolution and Structure, Oxford University Press, New York

Mytelka, Lynn Krieger (1999) Competition, Innovation and Competitiveness in Developing Countries, Development Centre, Organisation for Economic Co-operation and Development, Paris

Naughton, Barry, Editor (1997) The China Circle: Economics and Electronics in the PRC, Taiwan, and Hong Kong, Washington, D.C.: Brookings Institution Press

Negroponte, Nicholas (1995) Being Digital, Alfred A. Knopf, New York

National Software Alliance (NSA) (1998) Software Workers for the New Millennium: Global Competitiveness Hangs in the Balance, January, http://www.software-alliance.org

Nijhawan, Vinit (2000) Look Ma, No Wires, siliconindia.com, August 14th, http://www.siliconindia.com

OECD (1998a) The Software Sector: A Statistical Profile for Selected OECD Countries, Committee for Information, Computer and Communications Policy, January, http://www.oecd.org

OECD (1998b) Measuring Electronic Commerce: International Trade in Software, Committee for Information, Computer and Communications Policy, April, http://www.oecd.org

O'Riain, Sean (1999) "The Flexible Developmental State: Globalization, Information Technology and the 'Celtic Tiger'", Paper presented at the Global Networks, Innovation and Regional Development: The Informational Region as Development Strategy conference, November 11-13th, 1999, University of California, Santa Cruz, http://www2.ucsc.edu/cgirs/globalnet

Parthasarathy, Balaji (1999) "Institutional Embeddedness and Regional Industrialization: The State and the Indian Computer Software Industry", Paper presented at the Global Networks, Innovation and Regional Development: The Informational Region as Development Strategy conference, November 11-13th, 1999, University of California, Santa Cruz, http://www2.ucsc.edu/cgirs/globalnet

Paulk, Mark C., Bill Curtis, Mary Beth Chrissis and Charles V. Weber (1997) "Capability Maturity Model for Software", in Software Engineering: A Practitioner's Approach, Edited by M. Dorman and R.H. Thayer, McGraw-Hill, New York

Piore, Michael J. and Charles F. Sabel (1984) The Second Industrial Divide : Possibilities for Prosperity, Basic Books, New York

Pollard, Jane and Michael Storper (1996) "A Tale of Twelve Cities: Metropolitan Employment Change in Dynamic Industries in the 1980's," Economic Geography, v72, n1, page 1

Pressman, Roger S. (1997) “Software Engineering”, in Software Engineering: A Practitioner’s Approach, Edited by M. Dorman and R.H. Thayer, McGraw-Hill, New York

Putnam, Robert D. (1995) “Bowling Alone: America’s Declining Social Capital”, Current, Number 373, June

Rasiah, Rajah and Michael Best (2000) Industrial Transition in the Malaysian Electronics Industry, United Nations Industrial Development Organization, United Nations Development Programme

Raymond, Eric (1998) The Cathedral and the Bazaar, http://www.cse.ucsc.edu/

Raymond, Eric (1999) The Cathedral and the Bazaar: Musings on Linux and Open Source By an Accidental Revolutionary, O’Reilly, Sebastopol, California

Red Herring (Various)

Rhodes, Richard, Editor (1999) Visions of Technology: A Century of Vital Debate About Machines, Systems and the Human World, Simon & Schuster, New York

Ridley, Matt (1999) Genome: The Autobiography of a Species in 23 Chapters, Perrenial, New York

Rosenberg, Nathan (1982) Inside the Black Box: Technology and Economics, Cambridge University Press, New York

Rosenberg, Nathan (1994) Exploring the Black Box: Technology, Economics and History, Cambridge University Press, New York

San Jose Mercury News (Various)

Saxenian, AnnaLee (1994) Regional Advantage: Culture and Competition in Silicon Valley and Route 128, Cambridge, Mass.: Harvard University Press

Saxenian, AnnaLee (1999) Silicon Valley's New Immigrant Entrepreneurs, Public Policy Institute of California

Scott, Allen J. (1996) "Regional Motors of the Global Economy," Futures, v28, n5, page 391

Seitz, Frederick (1998) Electronic Genie: the Tangled History of Silicon, University of Illinois Press Urbana

Sheldrake, John (1996) Management Theory: from Taylorism to Japanization, International Thomson Business Press, Boston

Solow, Robert (1956) "A Contribution to the Theory of Economic Growth", Quarterly Journal of Economics, Volume 70, February

Solow, Robert (1957) "Technical Change and the Aggregate Production Function", Review of Economics and Statistics, Volume 34

Stanford Computer Industry Project (SCIP), http://www.stanford.edu/group/scip/

Steinmueller, W. Edward (1996) “The U.S. Software Industry: An Analysis and Interpretive History”, in The International Computer Software Industry: A Comparative Study of Industry Evolution and Structure, Edited by David C. Mowery, Oxford University Press, New York

Stephenson, Neal (1999) In the beginning ...was the command line, Avon Books, New York

Storper, Michael and Bennett Harrison (1990) Flexibility, Hierarchy and Regional Development : the Changing Structure of Industrial Production Systems and their forms of Governance in the 1990s, Los Angeles : Graduate School of Architecture and Urban Planning, University of California, Los Angeles

Sturgeon, Tim (1997a) "Turnkey Production Networks: A New American Model of Industrial Organization?", Working Paper 92A, Berkeley Roundtable on the International Economy, University of California, Berkeley, CA

Sturgeon, Tim (1997b) "Does Manufacturing Still Matter? The Organizational Delinking of Production from Innovation", Working Paper 92B, Berkeley Roundtable on the International Economy, University of California, Berkeley, CA

Tenner, Edward (1996) Why Things Bite Back: Technology and the Revenge of Unintended Consequences, Knopf, New York

Thayer, Richard H. (1997) Software Engineering Project Management, IEEE Computer Society, Los Alamitos, Calif.

Touraine, Alain (1995) Critique of Modernity, Blackwell, Oxford UK

Ullman, Ellen (1999) “The Myth of Order”, Wired, Version 7.04, April, http://www.wired.com

Wall Street Journal (Various)

Waring, Stephen P. (1991) Taylorism Transformed: Scientific Management Theory since 1945, University of North Carolina Press, Chapel Hill

Weber, Max (1968) Economy and Society; An Outline of Interpretive Sociology, New York, Bedminster Press

Weber, Max (1992) The Protestant Ethic and the Spirit of Capitalism, London:New York : Routledge

Weber, Steven (2000) “The Political Economy of Open Source Software”, BRIE Working Paper 140, E-Conomy Project Working Paper 15, UC Berkeley, http:www.e-conomy.berkeley.edu/

Webster, Frank (1995) Theories of the Information Society, Routledge, New York

Wired (Various)

Womack, James P., Daniel T. Jones and Daniel Roos (1991) The Machine that Changed the World, Harper Perennial, New York

World Bank (1993) The East Asian Miracle: Economic Growth and Public Policy, Oxford University Press, New York

World Bank (1997) World Development Report: The State in a Changing World, Oxford University Press, New York

World Bank (1998) World Development Report: Knowledge for Development, Oxford University Press, New York

Yeung, Henry Wai-Chung (1999) “Grounding Global Flows: Constructing an E-Commerce Hub in Singapore”, Paper presented at the Global Networks, Innovation and Regional Development: The Informational Region as Development Strategy conference, November 11-13th, 1999, University of California, Santa Cruz, http://www2.ucsc.edu/cgirs/globalnet