WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

# THE "FACTORY" APPROACH TO LARGE-SCALE SOFTWARE DEVELOPMENT: IMPLICATIONS FOR STRATEGY, TECHNOLOGY, AND STRUCTURE

Michael A. Cusumano

Revised
November 12, 1987

WP #1885-87

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

# THE "FACTORY" APPROACH TO LARGE-SCALE SOFTWARE DEVELOPMENT: IMPLICATIONS FOR STRATEGY, TECHNOLOGY, AND STRUCTURE

Michael A. Cusumano

Revised
November 12, 1987

WP #1885-87

Michael A. Cusumano
MIT Sloan School of Management
Sloan WP#1885-87 (Revised)

11/12/87
Software Project Paper #1

# THE "FACTORY" APPROACH TO LARGE-SCALE SOFTWARE DEVELOPMENT: IMPLICATIONS FOR STRATEGY, TECHNOLOGY, AND STRUCTURE

The impact of technology on organizational structure is a concern linking the fields of business history, strategy research, organizational theory, production and technology management, and comparative international studies. These diverse literatures have all treated a basic managerial dilemma resulting from two opposing demands: the desire of consumers for variety in product offerings; and the desire of managers responsible for engineering and production for standardization and control over processes, tools, and components as a way to reduce complexity and costs in product development and manufacturing, as well as in testing and customer service.

Laboratory or "job-shop" organizations have traditionally been used to construct new, highly complex, or "one-of-a-kind" products, but at high expense, due to the absence of economies of scale and scope. It has also been asserted that some product technologies "mature" stabilize over time, creating opportunities for firms to move toward more efficient means of engineering and production. The factory-type manufacturing organizations that may eventually emerge in an industry provide maximum efficiency in terms of costs per unit, but with little flexibility to introduce new products or processes quickly.

The basic argument of this paper and a series of accompanying case studies (Cusumano 1987b, c, d) is that organizational type and process choices -- such as job shops versus more rationalized factory environments, with standardized procedures, tools, and intermediate components -- may not be determined by the supposed characteristics of a technology or specific product types, or by the assumed level of industry "maturity." As firms accumulate experience or as changes in product features grow less frequent, managers may indeed build mass-production factories and compete on the basis of low cost and standardized designs. Newer factory concepts and computerized technologies also make it possible to create "flexible" factories that combine mass-production efficiency with job-shop flexibility in product variety.

Furthermore, some firms in an industry may choose to compete as job shops, making, for example, Rolls Royces instead of Model T cars. The passage of time may thus be important mainly in that experience or "industry maturity" brings with it a rise in the options firms have for their production strategies and organizations, with some moving from job shops to factories with varying levels of flexibility. A challenge for managers of relatively new and complex technologies that seem most approporate for job-shop production might then be how to use strategy, technology, and organizational structure to improve efficiency in product development and processing operations.

This paper treats large-scale software development environments for mainframes and minicomputers -- a segment of a technology that, compared to mass-produced products, is relatively new and highly complex due to the number of ways of implementing even similar functions, and the potential interactions among program components. The study asks a specific question: If one tries to measure structure by some simple indicators of inputs and process standardization and control, do producers of similar software products follow different process strategies -- for example, corresponding to job shops on one end of a hypothetical spectrum and more factory-like organizations on the other? The survey evidence presented here of major software facilities in the U.S. and Japan is that they do.

The conclusions follow that, if some firms create more factory-like production organizations even for software, then strategy and implementation are at least as important as the complexity of the technology in shaping the organization; and the emergence of factories need not be seen as simply a function of industry maturity. Both notions are consistent with Chandler's basic dictum that structure should follow strategy (Chandler 1962). It was hoped when designing this study that a set of criteria would make it possible to identify which firms seem more committed to disciplining software technology, and that studying these firms through detailed cases would then provide insights into how to develop and implement strategies for managing product and process development more effectively, especially for a relatively new and complex technology.

## THE PROBLEM FRAMED IN DIFFERENT LITERATURES

Business historians have long maintained that production organizations evolved over time and with accumulations of product and process knowledge, moving from craft-like job shops to mass-production factories. Most prominent is Chandler, who has described the evolution of factories during the 18th and 19th centuries in Britain, Europe and the United States, initially in the textile industry and then gun-making. This historical view has interpreted this movement as driven by managers attempting to raise worker productivity and lower unit costs by standardizing and then integrating production processes in a large, centralized facility; developing interchangeable components; closely coordinating the flow of each process; dividing and specializing labor; mechanizing or automating tasks; and imposing rigid accounting controls (Chandler 1977; Skinner 1985).

In the area of industrial organization theory, a school of thought rather complementary to Chandler has tended to see the characteristics of a particular technology, including inputs, processing, and outputs, as determining in large part how an organization evolves. Most important was Woodward (1965, 1970), who identified three basic types of production organizations, ranging from unit job shops and small-batch production, to large-batch and mass production, to continuous processing operations as in chemical manufacturing. The assumption here was that over time operations tended to become larger in scale and more complex, making it necessary for organizations to become larger and more complex, too, as they evolved from unit or batch operations to mass producers or continuous-production operations.

Somewhat in contrast is the contingency theory school associated with Lawrence and Lorsch (1967) as well as Galbraith (1973, 1977), which has argued there is no one best way to design the structure of an organization to manage a particular technology. What structure is most effective or appropriate depends on "what environmental demands or conditions confront the organization" (Scott, 1981: 208). A recent case study of medical-imaging scanner introduction similarly concluded that identical technologies can result in different structural outcomes, depending "on the specific historical process in which they are embedded" (Barley 1986: 107).

Production and operations management specialists such as Abernathy and Utterback (1975), Hayes and Wheelright (1979 and 1984), and Schmenner (1984) have also focused on the range of product and process options open to a firm as product technologies mature in a sort of life cycle. As with Chandler and Woodward, the underlying notion is that, as **product** innovations descrease over time, firms tend to take advantage of accumulated experience and innovate to rationalize their **process** technology and drastically reduce unit costs while standardizing quality, by moving from job-shop modes of production to large-scale engineering and factory manufacturing. The tradeoff in rigidity as a firm moved toward continuous production used to be extensive, because designs and processes became difficult and expensive to change. Indeed, the experience of Ford and its Model T production facilities in the 1920s demonstrates how a company can drive itself into near bankruptcy by pushing such strategic commitments too far -- for example, assuming product technology or consumer tastes were more stable than they were, and making factory systems so rigid they took long periods of time to change to new product or process technologies (Abernathy and Wayne 1974).

Recent developments in design and production technology have made it necessary to revise traditionally accepted tradeoffs between process flexibility and efficiency, as well as divisions between design and production. Rationalizing development processes without simply producing standardized goods or locking an organization into a single mode of production also combine product differentiation with process efficiency -- a rare but powerful combination of competitive skills (Porter 1980, 1985).

For example, during the 1950s and 1960s, Japanese auto firms pioneered "small-lot production" techniques, standardizing methods and inputs but not producing large lots of standardized final products due to machinery and workers capable of quickly changing to perform different operations or jobs (Schonberger 1982; Monden 1983; Cusumano 1985). Producers of machine tools, textile equipment, and various metal components have also managed to combine "small-lot" or job-shop flexibility in end-product variety with the productivity, quality, and management control of large factories (Jaikumar 1984 and 1986; Piore and Sabel 1984; Sabel 1987; Palframan 1987). Group technology concepts (putting together similar parts, problems, or tasks) have facilitated scheduling of parts production or arranging factory layouts as well as rationalizing product design and engineering, for small and large firms (Hyer and Wemmerloc 1984). Producers of semiconductors, like their counterparts in older industries like automobiles, routinely use standardized components and add end-process customization, increasing productivity while maintaining flexibility in design variety (Harvard Business School 1986). Computer-aided design tools integrated with flexible manufacturing systems (FMS) transfer digitalized designs automatically to manufacturing tools, allowing firms to automate this combining of modularized designs with new designs, with little or no penalty associated with low lot volumes (Skinner 1985; Jaikumar 1986; Meredith 1987).

Comparative international studies have observed the tendency of firms in certain countries to develop similar sets of strategies and structures. As noted, the small size of the domestic Japanese automobile market apparently persuaded managers to develop low-automation, flexible manufacturing systems during the 1950s and 1960s (Cusumano 1985). There appears to have been a similar trend in the Japanese machine tool industry (Jaikumar 1986).

A survey of 55 U.S. and 51 Japanese manufacturing plants found that the Japanese tended to establish similar types of manufacturing organizations, oriented toward more efficient, continuous-type processing operations (Lincoln, Hanada, and McBride 1986). Italy also seems to have produced a large number of firms, many of them textile machinery producers, emphasizing this combination of efficiency and specialization (Piore and Sabel 1984).

Difficult to separate are the effects of simply being from a certain culture or nation from the tendency of managers in that country to respond in kind to similar environmental conditions. Nonetheless, the question must still be answered to what degree managers operating in the same industry at the same time actually have and exercise options to manage product and process development. It also is not clear to what extent the peculiar characteristics of a technology might constrain the ability of managers to rationalize operations. These questions can be examined at least in part by seeing if firms producing similar software products do so with similar or different strategies and production organizations.

Determining why managers make the choices they do, or whether organizations evolve from deliberate decisions, is a complex subject of frequent and conflicting debate (Scott 1981). But, certainly, in designing new products and processes, there are options: Managers can refuse to worry about parts and procedures standardization, and encourage their engineers to design highly marketable products, which then might be mass-marketed. These firms might even be insensitive to development costs, if they could recoup large profits from mass sales, although, if they have a shortage of personnel, they may still want to maximize individual productivity. On the other hand, companies that choose to customize products might have even greater incentives to exploit similarities in tools, procedures, or parts across different product lines. Firms pursuing standardization (package) or customizing strategies, but especially the latter, might thus want to **standardize the development process and major components**, to reduce costs and perhaps improve quality as well.

This can be illustrated as follows. If a customer needs a product, whether it is an automobile, a machine tool, a semiconductor chip, or a software program, there are basically three options: obtain a fully customized product -- from a vendor or an in-house department; obtain a standardized or "packaged" product; obtain a semi-customized product (from a vendor or an in-house department that modifies a purchased standardized product). It follows that vendors should have three corresponding options: 1) sell a customized product; 2) sell a standardized product; 3) customize a semi-standardized product. Managers can adopt one of several strategies to manage product and process development.[1] Based on the fact that job shops continue to exist as factories appear, one can also draw a different type of product-process life cycle (Figure 1). With flexible factory models such as for machine tools or even software, moreover, it becomes difficult to separate product development from process development.

---

[1] A similar typology has recently been suggested independently by Lampel and Mintzberg 1987.

## Table 1: STRATEGIES FOR PRODUCT-PROCESS DEVELOPMENT

| MANUAL FULL CUSTOMIZATION: | IMPLEMENTATION: |
|---|---|
| Customize inputs and development process for each product | Maximize the capability of the organization to produce a unique product that will capture a high price from at least one customer |

Hardware Analogy: Job Shops
Software Analogy: Small Laboratory Environment

| MANUAL SEMI-CUSTOMIZATION: | IMPLEMENTATION: |
|---|---|
| Customize some inputs and processes and sell more than one of each product | Maximize the capability of the organization to produce a unique product that will capture a large share of the market |

Hardware Analogy: Batch Processing
Software Analogy: Large Development Center

| AUTOMATED FULL STANDARDIZATION: | IMPLEMENTATION: |
|---|---|
| Fully standardize inputs, processes and final products | Maximize the capability of the organization to produce a product with standard features at the lowest possible price |

Hardware Analogy: Model-T Factory
Software Analogy: Undesirable?

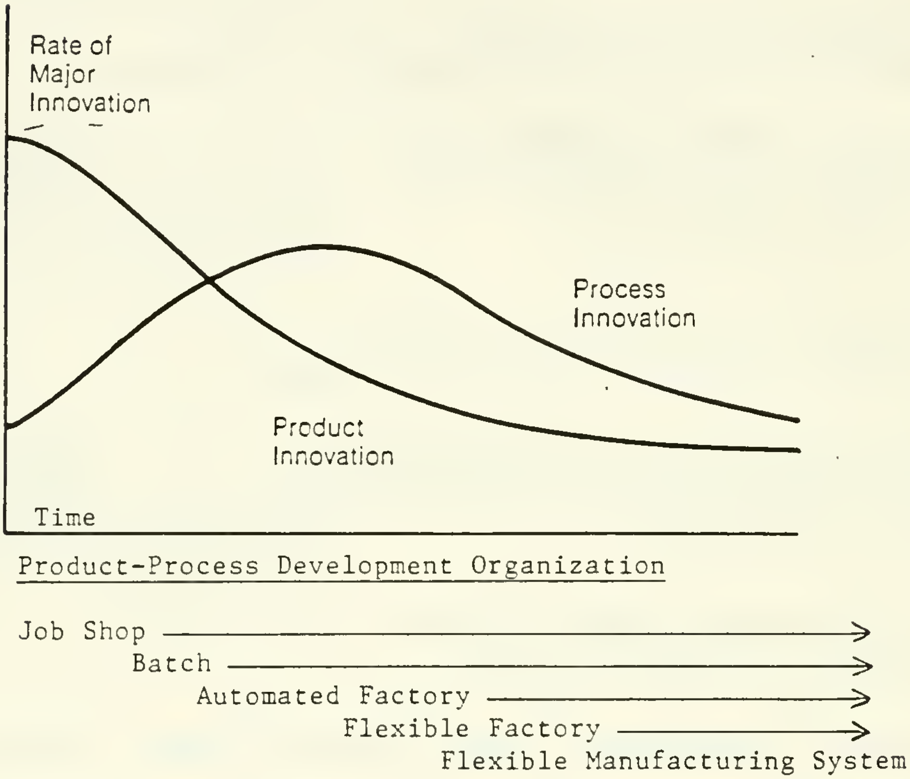
| MANUAL STANDARDIZED SEMI-CUSTOMIZATION: | IMPLEMENTATION: |
|---|---|
| Standardize inputs and processes but customize end products, with large-factory efficiency | Maximize the capability of the organization to produce semi-custom products at a low price through the use of as many standardized procedures and inputs as possible |

Hardware Analogy: Small-Lot Production, Group Technology
Software Analogy: Software Factory

| AUTOMATED STANDARDIZED CUSTOMIZATION: | IMPLEMENTATION: |
|---|---|
| Standardize inputs and processes but customize end products and automate processing | Maximize the capability of the organization to produce customized products at a low price through the use of highly flexible process techniques and/or automation |

Hardware Analogy: Automated Flexible Manufacturing Systems
Software Analogy: Program Generators, Used in Factories or Independently

FIGURE 1: REVISED PRODUCT PROCESS LIFE-CYCLE MATRIX



## FACTORY APPROACHES TO LARGE-SCALE SOFTWARE DEVELOPMENT

Software production dates back to the 1950s, when computers were first designed that could store in internal memory sets of instructions (programs) controlling basic operations and a variety of applications. As it has become refined over time, the series of phases followed in software development has come to resemble that for "hard" products, consisting of planning, detailed design, construction (coding), testing, and servicing (maintenance) phases. It typically takes years and hundreds of employees to develop and test programs such as operating systems for large mainframe computers or real-time control systems for factories or electric power plants. Furthermore, the clarity of designs, consistency of documentation, and lack or presence of defects or inadequate designs requiring extensive future modifications could impose enormous costs on software producers. Development costs for software programs over their lifetimes were typically about 10% for planning and design, less than 10% for coding, about 15% for testing, and as much as 70% for maintenance (Frank 1983).

As with hard manufacturing, it is difficult to generalize about software because there are numerous types of products for a wide variety of computers and applications. These fall into two basic categories: "systems" software, including operating systems, database management systems, telecommunications and other related programs designed to operate the basic functions of the computer or computer system; and "applications" programs, which operate a level above and implement specific functions like production control, payroll analysis, or word processing. Applications include "packaged" software -- products designed by producers for general sale --and customized software -- programs tailored to meet the unique needs of an individual customer.

Firms, or at least facilities, tend to specialize in particular types of software products, making it possible, theoretically, for them to achieve economies of scale or scope through standardization of tools, procedures, and inputs. In other industries, the rigidity imposed by such "rationalization," resulting in a reduced ability to adapt to changes in product or process technologies, or in market demands, has traditionally been seen as the major tradeoff a firm accepts in moving toward design standardization or factory mass-production. Software development may seem to be primarily a set of design and engineering activities for one-of-a-kind products, in the sense that companies make packages, systems software, or customized programs, with reproduction involving a simple process of replicating code. Thus, it might appear that software development organizations should all resemble job shops. Reinforcing this notion is a continuing debate among many programmers, managers, and academics over whether software development is more like an art or craft than an activity suitable for engineering or factory-like discipline and control (Brooks 1975; Shocman 1983; Hauptman 1986).

A problem with continuing to view a technology as a mere craft is that such attitudes may impede progress in rationalizing the design and production process. In software development, progress in improving productivity continues to be agonizingly slow. One study concluding around 1980 estimated that increases in programming output per man-hour had risen only about 5% annually since the 1960s, in contrast to improvements in computer processing capacity averaging about 40% a year (Horowitz and Munson 1984). Despite better computer languages, management methods, design techniques, and tools such as program generators that automatically produce code from high-level designs, demand for programmers in the mid-1980s outstripped the supply of programmers by about 10% in Japan and more than 20% in the U.S. (Zavala 1985; Aiso 1986). Observers began referring to this supply-demand imbalance as the "software crisis" as far back as 1969 (Hunke 1981; Frank 1983). In fact, U.S. companies desiring to buy fully customized applications programs typically had a 3- to 4-year wait (U.S. Department of Commerce 1984). In Japan, the typical backlog was 26.4 months (Kamijo 1986). Further hardware development, in large and small machines, continues to expand the demand for lengthy, complicated programs, even for personal computers, delaying for years the full utilization of microprocessors such as the Intel 80286 and 80386 (Businessweek 1987). In addition, a serious impediment to improvements in individual productivity are quality problems; data from IBM, TRW, and GTE indicate that fixing bugs in a completed program during operation can cost 100 times that of detecting errors in the design stage (Boehm 1976).

In short, the lack of programmers, rising demand for software, increasing length of programs needed to utilize improved hardware capacity, as well as the enormous impact of quality problems or product changes on productivity, have created a need to raise efficiency levels for software development. For a firm to increase its capability to produce a variety of high-quality products at lower costs than other companies, or simply to maximize its manpower and invested resources, should provide a potent competitive advantage -- in the software marketplace, as in other industries.

The idea of creating "software factories" to produce a variety of programs using standardized procedures, tools, and components reusable across different projects appears to have been first proposed by a Honeywell engineer at a 1968 NATO Science Conference on improving software productivity. Conference members criticized the idea as unworkable, largely due to three problems: One, it seemed too difficult to create program modules that would be efficient and reliable for all types of systems and which did not constrain the user. Two, it seemed impossible to write software that did not depend on specific characteristics of particular machines. And third, no method was then available to catalog program modules so they could be easily found and reused. (Horowitz and Munson 1984, citing McIlroy 1976; McNamara 1987).

These were and remain serious problems, although many large software producers have made progress in facilitating design and overall development efficiency, as well as reusability (Ramamoorthy 1984; Horowitz and Munson 1984; Goldberg 1986). Several companies and even the University of Southern California (Eliot and Scacchi 1986) even claim to have established software factories, and others have launched less elaborate software rationalization projects.

Of particular importance to this research project is an experiment that occurred at System Development Corporation (SDC), a producer of real-time software primarily for government contracts. By the mid-1970s, managers had encountered several common problems they hoped a more factory-like environment would solve: (1) Lack of discipline and repeatability or standardized approaches to the development process, with the result that SDC was continually reinventing products and processes, and not becoming as proficient at development or project control as managers wanted. (2) Lack of an effective way to visualize and control the production process, as well as to measure before the project was completed how well code implemented a design. 3) Difficulty in accurately specifying performance requirements before detailed design and coding, and recurrence of disagreements on the meaning of certain requirements, or changes demanded by the customer. 4) Lack of standardized design, management, and verification tools, making it necessary to reinvent these from project to project. 5) Little capability to reuse components, despite the fact that many application areas used similar logic and managers believed that extensive use of off-the-shelf software modules would significantly shorten the time required for software development. Several managers and development engineers decide to integrate a set of tools (program library, project databases, on-line interfaces between tools and databases, and automated support systems for verification, documentation, etc.) with standardized procedures and management policies for program design and implementation, and utilize this system in a centralized facility of about 200 programmers in Santa Monica, California. The concept and system of tools and methods SDC copyrighted under the name "The Software Factory" (Bratman and Court 1975 and 1977).

The SDC experiment was not particularly successful. Scheduling and budget accuracy improved dramatically, but the three problems raised at the NATO conference surfaced as well. For example, it was extremely difficult without portable computer languages to reuse code from one project on different computers and for different applications. This led to other problems. Most seriously, the tradition in SDC has been for project managers to create programming groups that would work at individual customer sites, in a sort of mobile job-shop mode of production. They did not like giving up control of development efforts to a centralized facility, and were not required by top management to use the Software Factory, leading to a decline in the flow of work into the facility. Ultimately, the company allowed project managers to remove programmers from the factory and it faded out of existence after approximately 10 projects. Programmers also tended to complain about unfamiliar, rigid standard, as well as the difficulty and inelegance of reusing other people's code.

In retrospect, it seems that SDC managers attempted to impose the factory infrastructure of standardized tools and methods, and reusability goals, on both project managers and programmers before software technology was refined enough to do this easily. Furthermore, architects of the factory failed to solve the matrix-management problems necessary to maintain a steady work flow into the new system. SDC gradually abandoned the effort by 1978, although it continued to use many of the factory procedures and some of the tools. The SDC model was also an important influence on the software standards later developed by the U.S. Department of Defense (Cusumano and Finnell 1987).

U.S. companies actively continued to develop better software tools, methods, and programming environments (Stucki and Walker 1981; Willis 1981; Boehm 1984; Howes 1984; Griffin 1984; Hoffnagle and Beregi 1986). This is the case even though SDC and other American firms no longer use terms such as "factories" and appear to prefer designations such as "laboratory" or "systems development center," or no label at all, to refer to their software organizations (McCue 1978; Hunke 1981). Some companies, such as IBM, also appear to have been slow too recognize software as a major part of their buisness, deserving of the same degree of attention, strategic planning, and process rationalization as their hardware operations. But another question is whether U.S. or software facilities in other countries have truly pursued a level of integration and standardization among people, systems, functions, tools, methods, and inputs sufficient to distinguish their operations from other large facilities with minimal standardization and integration.

## JAPANESE APPROACHES TO SOFTWARE ENGINEERING

Perhaps the most significant outcome of the SDC experiment was that reports from this company encouraged several Japanese software managers to pursue software factory organizations or at least greater efforts at process control and reusability (Iwamoto and Okada 1979; Matsumoto 1981; Mizuno 1985; Sakata 1985; Shibata 1985 and 1986). SDC did not introduce the factory concept into Japan, however. Japanese experimentation with centralized and highly disciplined programming environments actually predates the SDC experiment, going back to Hitachi's opening of the world's first facility called a software factory in 1969. NEC, Toshiba, and Fujitsu followed in 1976-1977, NT&T in 1985, and Mitsubishi Electric in 1987 (Table 2).

## Table 2: 1987 MAJOR JAPANESE SOFTWARE FACTORIES

Key: OS = Operating Systems
App = General Business Applications
Ind = Industrial Real-Time Control Applications
Tel = Telecommunications Software

Note: All facilities develop software for mainframes or minicomputers. Employee figures are 1987 estimates.

| Est. | Company | Facility/Project | Products | Employees |
|---|---|---|---|---|
| 1969 | Hitachi | Hitachi Software Works | OS, App | 1500 |
| 1976 | NEC | Software Strategy Project | | |
| | | (Fuchu) | OS | 2000 |
| | | (Mita) | App | 2500 |
| 1976 | Fujitsu | Numazu Factory | OS | 2000 |
| 1977 | Toshiba | Fuchu Software Factory | Ind | 2300 |
| 1977 | Fujitsu | Kamata Software Factory | App | 1500 |
| 1985 | Hitachi | Omori Software Works | App | 1500 |
| 1985 | NT&T | Software Development Div. (Mita and Shinagawa) | Tel | 400 |
| 1987 | Mitsubishi | Computer Factory | OS, App | 700 |

Source: Various company sources.

Various articles and reports from U.S. sources continue to find a more rationalized approach to software development in Japan than in the U.S. Four distinct trends seem to be emerging across several Japanese firms producing a variety of software products. One, is this construction of large, factory-like facilities to integrate tools, methods, and processes involved in software development. Second are attempts to exploit Japanese traditions of teamwork, discipline, and individual attention to quality by developing software tools and planning or reporting systems that facilitate group programming and a teamwork methodology throughout the software life cycle. Third are quality control techniques designed to catch bugs early, before they become difficult to fix. And fourth are extensive efforts to improve software quality and productivity through reusability of software modules and automation of software production (code generation) (Kim 1983; U.S. Department of Commerce 1984; Uttal 1984; Businessweek 1984; Johnson 1985; Haavind 1986).

If software followed the historical product-process life cycle and Japanese firms were advanced along these curves, this might explain why software factories seem to be so popular in Japan -- if indeed the Japanese facilities operated more like factories than large software facilities in countries such as the U.S. But Japanese firms began writing software in the late 1950s and early 1960s, several years behind U.S. counterparts, so more experience does not seem to be the explanation.

One U.S. group touring Japan concluded that, while the Japanese were not developing unique or more advanced software tools, they were using them more systematically then U.S. firms (Zelkowitz 1984). This suggests that the Japanese firms studied are simply more inclined than others toward makiing their production operations more efficient. A 1984 U.S. Department of Commerce report suggested that U.S. firms lag in software production management because Americans tend to view this technology more as a "craft" than the Japanese:

> The Japanese have...made impressive gains in the development of software tools and have encouraged their widespread use within their software factories to boost productivity...By contrast, while the United States is developing software engineering technology, the use of tools in U.S. firms is quite limited... Many U.S. software companies consider programming a craft and believe the future strength of the industry lies

> in its creativity rather than a disciplined approach to software development as do the Japanese."

A 1985 study by the <u>**Electronic Engineering Times**</u> cited culture as an explanation. It appeared to the writer in this journal that the Japanese were more effectively utilizing their traditional team or group approaches, as well as developing unique team-oriented software tools, in contrast to Americans, who, as usual, were overly dependent on small groups and highly skilled individuals:

> "[T]he approach to software technology taken by major developers in Japan, such as NEC, Fujitsu Ltd, and Hitachi Ltd., universally strive to harness that tradition of excellent teamwork... Each of these developers has automated versions of planning and reporting systems that enforce a strict teamwork methodology through the complete life cycle of a computer program -- from planning to design to maintenance, and without coding, since high-level language-source codes are automatically produced from the design documents. ... Until now, the Japanese have been hampered in their software development efforts by a lack of team-oriented tools. The tools borrowed from the United States simply do not fit the Japanese culture because they put too much control in too few hands.
>
> In America, industrial software development is generally done in groups that are as small as possible to minimize the communication problems among people. That makes the knowledge of each individual programmer a critical factor to the success of any software-development project. But...that is just not tolerable in the Japanese culture. As a consequence, the Japanese have had to perform basic research into software tools that can be wielded by many hands at once. Nobody else was going to develop group-programming tools for them."

If national differences do exist in the management of a particular technology, the reasons are no doubt complex. One explanation might be historical, such as a less prominent software craft culture ("hackers") in Japan, as opposed to the U.S., or simply more emphasis of Japanese companies on disciplined engineering and manufacturing practices. But a clear difference between Japan and the U.S. is the composition of their software markets, which might be encouraging certain Japanese firms to focus on process innovations necessary to rationalize design and production operations.

Although the figures are only approximate, in the U.S., about 60% of software sales in the early 1980s were standardized packages. In contrast, only about 5% of Japanese software sales were standardized packages; 95% involved some degree of customization (Table 3). One reason for this difference is that microcomputers accounted for only about 10% of Japanese software sales, compared to about 50% in the U.S. But Japanese customers also had a long history of preferring customized systems, placing tremendous demands on Japanese software producers faced with a shortage of skilled programmers and a backlog of orders, as indicated in Table 3.

Table 3: SOFTWARE MARKETS COMPARISON (1984-85)

| | Japan | USA |
|---|---|---|
| **Overall Market Characteristics:** | | |
| Total Market Revenues (Billion $) | 2.5 | 11 |
| Annual Demand Increases (%) | 25 | 25 |
| Annual Growth in Supply of Programmers (%) | 13 | 4 |
| Typical Wait for Customized Programs (Months) | 26 | 40 |
| **Product/Market Breakdown:** | | |
| Integrated Systems Software (%) | 5 | 16 |
| Package Software (%) | -- | 56 |
| Custom Software (%) | 95 | 28 |
| Microcomputer Software/Total Market (%) | 10 | 40 |
| All Systems Software | na | 70 |
| All Applications Software | na | 30 |
| **Computer Manufacturers as Suppliers of:** | | |
| All Systems Software (%) | 70 | 45 |
| All Applications Software (%) | 15 | 5 |

Sources: U.S. Department of Commerce 1984; Zavala 1985; Aiso 1986;

Kamijo 1986; **Businessweek** 1987.

Note: The figures cited are estimates by the author from data in the listed sources. Systems software includes operating systems, database management systems, telecommunications systems, and related programs. The size of the Japanese software market is underestimated because systems software is usually sold bundled with hardware.

The preceding discussion of product and process strategies, and of the large-scale software industry, raises several questions. One is do job shops, batch organizations, and factory-like facilities exist simultaneously in the software industry? A second question is which firms -- producing what specific products, and in what markets -- appear to resemble factories? If this is a Japanese trend, case studies of individual firms will be needed to examine why managers have chosen this strategy, and how they have implemented their factory systems. The final subject of this paper is to present the results of a survey of major Japanese and U.S. software producers.

## THE SURVEY

The published descriptions and stated objectives for the SDC Software Factory project (Bratman and Court 1975 and 1977) provided a basis for drawing up eight criteria to compare software facilities along a spectrum that should exist at least for large-scale engineering and manufacturing operations: process standardization and control; tool standardization and linkage; and inputs standardization and control (emphasis on reuse of software modules).

The factory-tool infrastructure the SDC project suggested consisted of a centralized program library to store modules, documentation, and completed programs; a central database to track production-management data; a uniform set of procedures for specification, design, coding, testing, and documentation; standardized project databases for groups working on different parts of a program; and an interface linking various tools and databases. These five variables constituted the core process and tool questions in the initial survey, which attempted to see if managers in the U.S. and Japan were currently emphasizing a similar type of infrastructure. In addition, since an objective of the factory strategy was to produce standardized components for reuse, rather than "reinvent the wheel" with every customer order, three questions were included about reusability.

Major software producers in Japan and the U.S. were identified through literature surveys and lists of software producers, and sent a questionnaire containing eight core questions plus more than a dozen others, many of an experimental nature, to provide supplementary data.[2] Optional questions also requested performance measures such as actual rates of reused code in a

---

[2] Additional questions were also sent to survey participants, although comments from the respondees, site visits and interviews, as well as partial correlation analysis, revealed that many of the non-core questions were not particularly useful for measuring "rationalization" along large-scale engineering and manufacturing lines. For example, three questions asked for emphasis on standardization of languages for high-level design, module description, and coding. It turned out that Japanese and English were mainly used for high-level design, and many managers did not know how to answer; Japanese tended to develop specialized languages for module description because they were less comfortable than U.S. programmers in using English-based languages for this purpose, which made it unfair to U.S. firms to use this question; and coding languages were often determined by customers. A question about top-down design was discarded because emphasis on this tended to contrast with a more factory-type process of combining new and old code in layers. Similarly, questions about emphasis on high-level abstraction or layering were discarded because not everyone knew how to interpret these.

recent sample year. For the core questions, managers either responsible for overall software engineering management or with sufficient experience to present an overview of practices for the entire facility were asked to rank the emphasis of themselves and general managerial policy at their facilities on a scale of 0 to 4 (see Table 4), as well as to comment on each answer. Questionnaires were directed to managers at the facility or product-area level, since software practices usually differed significantly among divisions in diversified or large firms, and some diversity seemed useful to meet different market or internal needs.

The sample was limited to facilities or departments making products that usually require large amounts of people, time, and tools to develop, and which might therefore provide incentives for managers at least on the facility level to seek similarities and common components or tools across different projects: operating systems for mainframes or minicomputers ("systems" software); and real-time applications programs, such as for factory control or reservations systems ("applications" software). These were further broken down into telecommunications software (applications and systems were combined because of the smallness of the sub-sample); commercial operating systems; industrial operating systems; real-time control applications; and general business applications.

All the Japanese firms contacted filled out the survey; about 75% of U.S. firms contacted completed the survey. To check answers, two managers at each firm or facility were asked to respond. About half the companies returned two completed surveys for each type of facility; the answers were extremely similar, usually differing by only a few percentage points, and were averaged. Other answers represent single responses.[3]

A factor analysis procedure with varimax rotation indicated that the eight questions constituted three orthogonal factors with eigenvalues rounding to approximately 1.0 (actual 2.91, 0.88, and 0.67); these explained 96.1% of the variance in survey answers. For each factor, the variables with a strong loading (approximately 0.51 to 0.78) were summed and used to test differences in the average Japanese and U.S. scores, as well as to test if product type or country origin of the facility were significantly correlated with the process and reuse scores.[4]

The data reported in Table 5 reflects scores for each dimension and the total for eight variables on a basis of 100%, with maximum scores of 4 for each variable. Table 6 compares the average Japanese and U.S. responses to the process, tools, and inputs dimensions. Table 7 summarizes the results of one-way analysis of variance tests to determine the effects of product types or country of origin on the scores reported for the three dimensions. Tables 8 through 10 compare actual reuse rates reported by the Japanese and U.S. facilities, and analyze correlations with the three survey dimensions as well as types of products and country of origin.

Since the sample size is relatively small in absolute numbers, the results of this analysis must be considered as no more than suggestive of

---

[3] In the case of Toshiba, a single large facility (approximately 2500 programmers) had different departments producing both systems and applications programs using identical procedures and tools, and the manager responsible for technical development, Dr. Yoshihiro Matsumoto, submitted one set of answers and asked that they be counted twice, under both systems and applications facilities.

[4] This procedure is recommended as a simple data reduction technique by Comrey 1973 and Tabachnick and Fidell 1983. Comrey suggested that loadings of .55 (explaining 30% of the variance) were "very good," and .63 (40% variance) or over "excellent."

patterns existing at software facilities in the U.S. and Japan. It should be noted, however, that the surveyed Japanese firms account for the vast majority of software written and sold in Japan, and the surveyed U.S. firms include most of the largest producers of computer operating systems, applications software, and related services such as data bases, which also have a large software component.[5]

## Table 4: SURVEY AND SAMPLE OUTLINE

**SAMPLE:** n = 44 (23 Japanese, 21 U.S.)

**SURVEY PARTICIPANTS:** Software Development Managers

**ANSWERS KEY:**
4 = Capability or policy is FULLY USED OR ENFORCED
3 = Capability or policy is FREQUENTLY USED OR ENFORCED
2 = Capability or policy is SOMETIMES USED OR ENFORCED
1 = Capability or policy is SELDOM USED OR ENFORCED
0 = Capability or policy is NOT USED

---

[5] The top three Japanese firms ranked by software sales in 1986 were NEC ($507 million), Fujitsu ($389 million), and Hitachi ($331 million). NEC ranked fourth in the world, behind IBM ($5,514 million), Unisys ($861), and DEC ($560). The Japanese sales figures considerably understate actual software development, because Japanese firms included ("bundled") systems software with mainframe and minicomputer hardware prices, although the size of systems software operations corresponds roughly to hardware sales. The largest Japanese producers of mainframes by 1986 sales were Fujitsu ($2,470 million), NEC ($2,275), Hitachi ($1,371), and Mitsubishi ($185); the largest sellers of minicomputers were Toshiba ($766), Fujitsu ($620), and Mitsubishi ($475). On the U.S. side, IBM was by far the world's largest producer of hardware and software; three of its facilities are represented in the survey. Unisys, which ranked 2nd in world software sales, has two facilities in the survey. In services, TRW ranked 1st and General Motors/EDS 3rd; Control Data, Martin Marietta, and NT&T 6th, 7th, and 8th; Boeing and IBM 12th and 13th (Datamation 1987). Other large Japanese producers of software included in this survey were subsidiaries of Hitachi and NEC, including Nippon Business Consultants and Hitachi Software Engineering (Hitachi), as well as NEC Software, NEC Information Systems, and Nippon Electronics Development (NEC) (Kiriu 1986).

## SURVEY QUESTIONS:

### Process Standardization and Control (Score of 12 = 100%)

1) A centralized program library system to store modules and documentation.

2) A central production or development data base connecting programming groups working on a single product family to track information on milestones, task completion, resources, and system components, to facilitate overall project control and to serve as a data source for statistics on programmer productivity, costs, scheduling accuracy, etc.

3) A uniform set of specification, design, coding, testing, and documentation procedures used among project groups within a centralized facility or across different sites working on the same product family to facilitate standardization of practices and/or division of labor for programming tasks and related activities.

### Tool Standardization and Interface (Score of 8 = 100%)

4) Project data bases standardized for all groups working on the same product components, to support consistency in building of program modules, configuration management, documentation, maintenance, and potential reusability of code.

5) A system interface providing the capability to link support tools, project data bases, the centralized production data base and program libraries.

### Inputs Standardization and Control (Score of 12 = 100%)

6) Formal management promotion (beyond the discretion of individual project managers) that new code be written in modular form with the intention that modules (in addition to common subroutines) will then serve as reusable "units of production" in future projects

7) Formal management promotion (beyond the discretion of individual project managers) that, if a module designed to perform a specific function (in addition to common subroutines) is in the program library system, rather than duplicating such a module, it should be reused.

8) Monitoring of how much code is being reused

### Total for 8 Variables (Score of 32 = 100%)

## Table 5: SUMMARY AND RANKING OF SURVEY SCORES (%)

Note: Japanese Facilities indicated by *

| COMPANY/FACILITY | Process | Tools | Inputs | Total |
|---|---|---|---|---|
| **Telecommunications Software** | | | | |
| *NEC Switching Systems | 100.0 | 93.8 | 100.0 | 98.4 |
| *NT&T Applications | 75.0 | 87.5 | 91.7 | 84.4 |
| *Mitsubishi Electric | 83.3 | 87.5 | 75.0 | 81.3 |
| AT&T Bell Labs Applications) | 83.3 | 75.0 | 58.3 | 71.9 |
| *Hitachi Totsuka Works | 83.3 | 37.5 | 50.0 | 59.4 |
| *NT&T Systems | 66.7 | 50.0 | 50.0 | 56.3 |
| **Commercial Operating Systems** | | | | |
| *NEC Fuchu Factory | 95.8 | 87.5 | 95.8 | 93.8 |
| *NEC Software, Ltd. | 91.7 | 87.5 | 83.3 | 87.5 |
| IBM-Endicott | 83.3 | 87.5 | 66.7 | 78.1 |
| Control Data | 83.3 | 93.8 | 58.3 | 76.6 |
| *Hitachi Software Works | 91.7 | 50.0 | 66.7 | 71.9 |
| *Fujitsu Numazu Factory | 79.2 | 81.3 | 58.3 | 71.9 |
| *Mitsubishi Electric | 91.7 | 12.5 | 58.3 | 59.4 |
| IBM-Raleigh | 100.0 | 62.5 | 16.7 | 59.4 |
| Data General | 62.5 | 75.0 | 41.7 | 57.8 |
| Sperry/Unisys | 54.2 | 68.8 | 45.8 | 54.7 |
| **Industrial Operating Systems** | | | | |
| *Toshiba Software Factory | 83.3 | 75.0 | 100.0 | 87.5 |
| Boeing | 75.0 | 75.0 | 25.0 | 56.3 |
| **Real-Time Control Applications** | | | | |
| TRW | 100.0 | 100.0 | 75.0 | 90.6 |
| *Toshiba Software Factory | 83.3 | 75.0 | 100.0 | 87.5 |
| Sperry/Unisys | 100.0 | 100.0 | 66.7 | 87.5 |
| *Hitachi Omika Works | 75.0 | 75.0 | 83.3 | 78.1 |
| SDC/Unisys | 58.3 | 87.5 | 91.7 | 78.1 |
| *Mitsubishi Electric | 83.3 | 62.5 | 66.7 | 71.9 |
| Hughes Aircraft | 79.2 | 93.8 | 45.8 | 70.3 |
| Boeing | 83.3 | 75.0 | 25.0 | 59.4 |
| Honeywell | 75.0 | 12.5 | 16.7 | 37.5 |
| Draper Laboratories | 37.5 | 25.0 | 8.3 | 23.4 |

Table 5 Continued

| COMPANY/FACILITY | Process | Tools | Inputs | Total |
|---|---|---|---|---|
| **General Applications** | | | | |
| *NEC Mita | 95.8 | 87.5 | 89.6 | 91.4 |
| *NEC Information Services | 91.7 | 100.0 | 75.0 | 87.5 |
| Control Data | 83.3 | 100.0 | 75.0 | 84.4 |
| *Nippon Systemware | 75.0 | 62.5 | 91.7 | 78.1 |
| *Fujitsu Kamata Software Factory | 75.0 | 75.0 | 79.2 | 76.6 |
| *Hitachi Omori Works | 83.3 | 62.5 | 66.7 | 71.9 |
| IBM (Office Products) | 75.0 | 87.5 | 58.3 | 71.9 |
| Martin Marietta/MD | 75.0 | 50.0 | 79.2 | 70.3 |
| *Nippon Business Consultants | 58.3 | 50.0 | 83.3 | 65.6 |
| Cullinet | 62.6 | 68.8 | 48.6 | 58.9 |
| EDS/GM | 62.5 | 62.5 | 50.0 | 57.8 |
| Martin Marietta/Denver | 83.3 | 50.0 | 25.0 | 53.1 |
| *Hitachi Software Engineering | 62.5 | 18.8 | 62.5 | 51.6 |
| *Mitsubishi Electric | 83.3 | 0 | 33.3 | 43.8 |
| *Nippon Electronics Development | 58.3 | 0 | 50.0 | 40.6 |
| Computervision | 20.8 | 43.8 | 25.0 | 28.1 |

## Table 6: COMPARISON OF AVERAGE JAPANESE AND U.S. SURVEY SCORES

| Dimension | n = 23<br>Japanese | n = 21<br>U.S. | t-value |
|---|---|---|---|
| Process (Std. Dev.) | 81.2 (11.7) | 73.2 (19.7) | 1.64* |
| Tools (Std. Dev.) | 61.7 (30.0) | 71.1 (24.2) | -1.14** |
| Inputs (Std. Dev.) | 74.4 (18.9) | 47.8 (23.5) | 4.16* |
| Total (Std. Dev.) | 73.7 (15.9) | 63.1 (18.0) | 2.07*** |

---

* $p < .01$
** $p < .10$
*** $p < .05$

## Table 7: EFFECTS OF COUNTRY AND PRODUCT TYPE

Test: ONE WAY ANALYSIS OF VARIANCE
n = 44

A. Effects on Process Score:

| Variable | F-ratio | Sig. Level |
|---|---|---|
| Country# | 2.698 | .1080 |
| Product Type## | .948 | .4465 |

B. Effects on Tools Score

| Variable | F-ratio | Sig. Level |
|---|---|---|
| Country# | 1.304 | .2599 |
| Product Type## | .630 | .6444 |

C. Effects on Inputs Score

| Variable | F-ratio | Sig. Level |
|---|---|---|
| Country# | 17.296 | .0002 |
| Product Type## | .267 | .8974 |

---

\# Coded as 0 = Japanese facility, 1 = U.S. facility

\#\# Coded as 1 = Telecommunications Software, 2 = Commercial Operating Systems, 3 = Industrial Operating Systems, 4 = Real-Time Control Applications, 5 = General Applications

## Table 8: COMPARISON OF REPORTED JAPANESE AND U.S. REUSE RATES

| n = 13<br>Japanese (Std. D.) | n = 15<br>U.S. (Std. D) | t-value |
|---|---|---|
| 33.8% (14.5) | 14.3 (14.0) | 3.64* |

---

* p < .01

## Table 9: MULTIPLE REGRESSION MODEL FOR REUSE RATES

n = 28

| Ind. Variable | Coefficient | t-value | Sig. Level |
|---|---|---|---|
| Constant | 3.94 | 0.30 | 0.77 |
| Process | 0.03 | 0.19 | 0.85 |
| Tools | -0.06 | -0.52 | 0.61 |
| Inputs | 0.37 | 2.96 | 0.01 |

Adj. R-Sq. = 0.20
F-Ratio = 3.31
P-value = 0.04

## Table 10: EFFECTS OF COUNTRY AND PRODUCT TYPE ON REUSE RATES

Test: ONE WAY ANALYSIS OF VARIANCE
n = 28

| Variable | F-ratio | Sig. Level |
|---|---|---|
| Country# | 13.215 | .0012 |
| Product Type## | .692 | .6052 |

---

# Coded as 0 = Japanese facility, 1 = U.S. facility

## Coded as 1 = Telecommunications Software, 2 = Commercial Operating Systems, 3 = Industrial Operating Systems, 4 = Real-Time Control Applications, 5 = General Applications

## DISCUSSION

The survey results appear to support two hypotheses. One is that there are clearly variations in emphasis among software managers, regarding the strategies and structures of their organizations. Despite potential views of software development as largely a craft, art, or "job-shop" type of operation, some managers at facilities making similar types of products clearly placed more emphasis on control and standardization of processes, basic tools, and reusable inputs (modules of code).

Within the same product types, for example, total scores ranged from 56.3% to 98.4% in Telecommunications Software; 54.7% to 93.8% in Commercial Operating Systems; 56.3% to 87.5% in Industrial Operating Systems; 23.4% to 90.6% in Real-Time Control Applications; and 28.1% to 91.4% in General Business Applications (Table 5). It may be difficult to distinguish facilities ranking within a few percentage points; but companies on opposite ends of the spectrums that emerged from the survey rankings must be different and in instructive ways. Moreover, the analysis of variance tests confirmed that product types as defined in this paper had no significant impact on where managers scored on either dimension studied.

A second hypothesis one might generate from the discussion of Japanese approaches to software engineering is that there are some national differences between the U.S. and Japan in management emphasis, at least as far as was measurable in this limited survey. The data provide some support for this. Among the three dimensions studied, the t-tests indicated that only the average responses for the inputs variables -- 74.4% for the Japanese and 47.8% for the U.S. -- and the total scores -- 73.7% for the Japanese and 63.1% for the U.S. -- were significantly different (Table 6). This does suggest that factory-type approaches centering on process standardization and control as well as reusability might be more commonly emphasized in Japan. Reported Japanese reuse rates were also significantly higher than U.S. rates (33.8% to 14.3%) (Table 8). Japanese scores also tended to be higher on the process variables, and U.S. scores higher for the tools variables, although neither averages by themselves were significantly different at a confidence interval even of 90%.

The only dimension that correlated significantly with reported reuse rates was emphasis on inputs reuse (Table 9). That standardization of processes and tools were not significant suggests that reusability is a complex phenomenon and probably has much to do with the similarity of work flowing through a particular facility. Nonetheless, the analysis of variance tests confirmed that product type was not significant, while country of origin significantly correlated with the inputs scores and reported reuse rates (Tables 7 and 10). This data suggests that Japanese applications producers, who clearly are marketing customized products, as well as Japanese systems producers, who sell basic software, both tend to emphasize reusability. In the U.S., System Development Corp./Unisys, and to a lesser extent Martin Marietta/Maryland and TRW, also appeared to be following a reusability or customizing strategy. But, as noted earlier, particular features of the market in Japan, which almost universally has demanded customized products, perhaps encouraged firms in this country more than in the U.S. to pursue standardization and customization relying on the construction of reusable components.

## IMPLICATIONS

The survey was a "first cut" attempt to measure differences in emphases among managers at major software facilities in the U.S. and Japan. This identified firms across a spectrum resembling flexible factories on the upper end and job shops on the lower end, for a variety of product types. The real value from this research should come from detailed, historical case studies of firms at the higher end of the spectrum, which should show how it is possible to impose "factory" discipline over engineering and production operations even for a relatively new and complex technology.

Over time, if not at the present, facilities at the upper end of the spectrum may become able to produce customized (or semi-customized) products similar in performance to those products (packaged or customized) made by firms at the lower end of the spectrum but at a lower cost, due to savings from process management or elimination of having to "reinvent" components. This may provide an important competitive advantage, as it has in industries such as semiconductors, machine tools, and even automobiles. Managers of product development and production need to ask themselves if they are doing all they can to improve their operations. If some firms deemphasize standardization, integration of tools and processes, and reuse of components, while focusing essentially on the individual engineer, the individual tool, or the final product, then they may not be fully developing -- that is, compared to some of their competitors -- **organizational capabilities** to maximize the performance of their technical people and invested resources.

All such "rationalization" of product and process development depends, at least to some extent, on the nature of the marget segments a firm wishes to serve. The question then becomes, within those segments, is it possible to be more efficient -- for example, semi-customizing products from reused components rather than building all programs from scratch, or simply investing in tool and process development and then standardizing around technologies that seem to work best. The scale of the facility should be less important than the degree of standardization of processes and tools, integration, or reuse rates, although scale may be important to justify the financial investment process development.

For software, a lingering issue is to what extent the design complexity of this technology can ever be reduced. Previous innovations leading to improvements in programming productivity include high-level languages, time sharing, integrated programming environments such as Unix work, new programming techniques, expert systems and artificial intelligence-based tools, graphics tools, automated programming, testing automation, high-powered workstations, and rapid prototyping techniques (Brooks 1987). All of these are being used and continually developed as basic tools in software factories (Cusumano 1987b, 1987c, 1987d), although these can probably be used with equal effectiveness in job shops or laboratories -- if they are applied consistently.

A larger issue is technological change and managerial influence. A shift in focus from the individual and individual tools and techniques, to the organization and process management -- reflects a movement one might expect with any product and process as firms accumulate experience in design and production, and as market demands become better defined, at least compared to the earliest days of an industry. But the software example suggests this is not a movement necessarily constrained by the nature of a technology or by the simple passage of time. The critical variables may be managerial strategy and efforts at implementation, although further discussion of this topic must await additional research on individual firms.

## APPENDIX TABLES

### VARIMAX ROTATED FACTOR MATRIX

| Variable/Factor | Inputs | Process | Tools |
|---|---|---|---|
| library | -0.20667 | **0.51709** | 0.35414 |
| central data base | 0.27622 | **0.65561** | 0.11498 |
| uniformity | 0.11185 | **0.65900** | 0.13972 |
| project data base | 0.17609 | 0.13763 | **0.77652** |
| interface | 0.29765 | 0.23977 | **0.62033** |
| design for reuse | **0.70171** | 0.12273 | 0.48046 |
| reuse promotion | **0.74770** | 0.08013 | 0.16469 |
| monitoring reuse | **0.51412** | 0.49757 | 0.05278 |

### EIGENVALUES AND PERCENT OF VARIANCE

| Variable | Factor | Eigenvalue | % Variance | Cum. Percent |
|---|---|---|---|---|
| library | 1 | 2.90787 | 62.8 | 62.8 |
| central database | 2 | .87607 | 18.9 | 81.7 |
| project database | 3 | .66664 | 14.4 | 96.1 |
| interface | 4 | .15759 | 3.4 | 99.5 |
| uniformity | 5 | .02104 | .5 | 100.0 |
| design for reuse | 6 | -.09701 | .0 | 100.0 |
| reuse promotion | 7 | -.16238 | .0 | 100.0 |
| monitoring reuse | 8 | -.30270 | .0 | 100.0 |

## BIBLIOGRAPHY

ABERNATHY, William J. and James UTTERBACK, "Dynamic Model of Process and Product Innovation," Omega, 3 (1975), 639-657.

ABERNATHY, William J. and Kenneth WAYNE, "Limits of the Learning Curve," Harvard Business Review (September-October 1974), 109-119.

AISO, H. (Keio University), "Overview of Japanese National Projects in Information Technology," International Symposium on Computer Architecture, Lecture 1, 2 June 1986, Tokyo.

BARLEY, Stephen R., "Technology as an Occasion for Structuring: Evidence from Observations of CT Scanners and the Social Order of Radiology Departments," Administrative Science Quarterly, 31 (1986): 78-108.

BOEHM, Barry W., "Software Engineering," IEEE Transactions on Computers, C-25 (1976), 1126-1141.

_____, "A Software Development Environment for Improving Productivity," Computer (June 1984), 30-44.

BRATMAN, Harvey, and Terry COURT (System Development Corporation), "The Software Factory," Computer (May 1975), 28-37.

____, "Elements of the Software Factory: Standards, Procedures, and Tools," in Infotech International Ltd., Software Engineering Techniques, Berkshire, England, Infotech International Ltd. 1977, 117-143.

BROOKS, Frederick P. Jr., The Mythical Man-Month: Essays in Software Engineering, Reading, Ma., Addison-Wesley, 1975.

BROOKS, Frederick P. Jr., "Essence and Accidents of Software Engineering," Computer, April 1987, 10-19.

BUSINESSWEEK, "Japan's Push to Write 'World-Class' Software," 27 February 1984, 96-98.

BUSINESSWEEK, "The Free-for-All Has Begun," 11 May 1987, 148-159.

CHANDLER, Alfred D. Jr., Strategy and Structure: Chapters in the History of American Industrial Enterprise, Cambridge, MA, MIT Press, 1962.

____, The Visible Hand: The Managerial Revolution in American Business, Cambridge, M.A., Harvard University Press, 1977.

COMREY, A. L., A First Course in Factor Analysis, New York, Academic Press, 1973.

CUSUMANO, Michael A., The Japanese Automobile Industry: Technology and Management at Nissan and Toyota, Cambridge, MA, Harvard University Press, 1985.

CUSUMANO, Michael A. and David E. FINNELL, "A U.S. "Software Factory" Experiment: System Development Corporation" **M.I.T Sloan School of Management**, Working Paper #1887-87, 1987.

CUSUMANO, Michael A., "Hitachi: Pioneering a 'Factory' Strategy and Structure for Large-Scale Software Development," **M.I.T Sloan School of Management**, Working Paper #1886-87, 1987b.

CUSUMANO, Michael A., "Toshiba's Fuchu Software Factory: Strategy, Technology, and Organization," **M.I.T Sloan School of Management**, Working Paper #1939-87, 1987c.

CUSUMANO, Michael A., "NEC: Standardization Strategy for a Distributed 'Software Factory' Structure," **M.I.T Sloan School of Management**, Working Paper, 1987d.

DATAMATION, 15 June 1987, 28-32.

ELIOT, Lance B., and Walt SCACCHI, "Towards a Knowledge-Based System Factory," IEEE Expert, Winter 1986, pp. 51-58.

FRANK, Werner L., **Critical Issues in Software**, New York, John Wiley & Sons, 1983.

FUJINO, Kiichi (Vice-President, NEC), "Software Development for Computers and Communications at NEC," **Computer** (November 1984,

____, Interviews, 7/28/86 and 9/8/87.

GALBREATH, Jay, **Designing Complex Organizations**, Reading, MA, Addison-Wesley, 1973.

____, **Organization Design**, Reading, MA, Addison-Wesley, 1977.

GOLDBERG, R., "Software Engineering: An Emerging Discipline," **IBM Systems Journal** 25 (1986), 334-353.

GRIFFIN, William G., "Software Engineering in GTE," **Computer** (November 1984), 66-72.

HAAVIND, Robert, "Tools for Compatibility, " **High Technology** (August 1986), 34-42.

HARVARD BUSINESS SCHOOL, "VLSI Technology, Inc. (A)," Case Study 0-686-128 (1986).

HAUPTMAN, Oscar, "Influence of Task Type on the Relationship Between Communication and Performance: The Case of Software Development," **R&D Management** 16 (1986), 127-139.

HAYES, Robert H. and Steven C. WHEELRIGHT, "Link Manufacturing Process and Product Life Cycles," Harvard Business Review (January-February 1979), 133-140.

____, Restoring Our Competitive Edge: Competing through Manufacturing, New York, John Wiley & Sons, 1984.

HITACHI SEISAKUSHO KABUSHIKI KAISHA (Hitachi Ltd.), Kanagawa kojo 15 nen no ayumi (15-year history of the Kanagawa Works), Hitachi Ltd., Kanagawa, 1979.

HOFFNAGLE, G.F., and W.E. BEREGI, "Automating the Software Development Process," IBM Systems Journal, 24 (1985).

HOROWITZ, Ellis and John B. MUNSON, "An Expansive View of Reusable Software," IEEE Transactions on Software Engineering, SE-10 (1984), 477-486.

HOWES, Norman R., "Managing Software Development Projects for Maximum Productivity," IEEE Transactions on Software Engineering, SE-10 (1984), 27-49.

HUNKE, H., ed., Software Engineering Environments, Amsterdam, North-Holland, 1981.

HYER, Nancy L. and Urban WEMMERLOC, "Group Technology and Productivity," Harvard Business Review (July-August 1984), 140-149.

IWAMOTO, Kanji and Masashi OKADA (NEC), "Purojekuto kanri no tsuru" (Project control tools), Joho shori (Information processing), 20 (1979).

JAIKUMAR, Ramchandran, "Flexible Manufacturing Systems: A Managerial Perspective," Harvard Business School Working Paper #1-784-078 (January 1984).

____, "Postindustrial Manufacturing," Harvard Business Review (November-December 1986), 301-308.

JOHNSON, R. Colin, "Tools for Software Factory," Electronic Engineering Times (11 February 1985), 1, 63.

KAMIJO, Fumihiko, "Information Technology Activities in the Japanese Software Industry," Oxford Surveys in Information Technology, Vol. 3, 1-35 (1986).

KIM, K.H., "A Look at Japan's Development of Software Engineering Technology," Computer (May 1983), 26-37.

KINNUKAN, Paul, "Flexible Systems Invade the Factory," High Technology (July 1983), 32-43.

KIRIU, Hiroshi, Sofutouea sanyo no jitsuzo (The status of the software industry), Tokyo, Nikkan Shobo, 1986.

LAMPEL, Joseph, and Henry MINTZBERG. "Customizing Strategies...and Strategic Management," McGill University Working Paper, March 1987.

LAWRENCE, Paul R., and Jay W. LORSCH, **Organization and Environment: Managing Differentiation and Integration**, Boston, Harvard Business School, 1967.

LINCOLN, James R., Mitsuyo HANADA, and Kerry MCBRIDE, "Organizational Structures in Japanese and U.S. Manufacturing," **Administrative Science Quarterly**, 31 (1986): 338-364.

MATSUMOTO Yoshihiro (Toshiba), "Management of Industrial Software Production," **Computer** (February 1984), 59-71.

_____, "A Software Factory: An Overall Approach to Software Production" in Peter Freeman, ed., **Software Reusability** (IEEE Tutorial, 1987).

_____, et al., "SWB System: A Software Factory," in H. Hunke, ed., **Software Engineering Environments**, Amsterdam, North-Holland, 1981.

McCUE, G.H. "IBM'S Santa Teresa Laboratory: Architectural Design for Program Development," **IBM Systems Journal**, 17 (1978),

McILROY, M.D., "Mass-Produced Software Components," in J.M. Buxton, P. Naur, and B. Randell, Eds., **Software Engineering Techniques: 1968 NATO Conference on Software Engineering**, 1976.

McNAMARA, Donald (General Electric), "Software Factories." Lecture, Wang Institute of Graduate Studies, Lowell, MA, 2 February 1987.

MEREDITH, J., "The Strategic Advantages of New Manufacturing Technologies for Small Firms," **Strategic Management Journal**, 8 (1987), 249-258.

MIZUNO, Yukio (Senior Vice-President, NEC Corporation). Interview 9/26/85.

MONDEN, Yasuhiro, **Toyota Production System**, Norcross, Ga., Industrial Engineering and Management Press, 1983.

MURAKAMI, Noritoshi, Manager, Software Development Planning Office, Fujitsu, Ltd. Interview, 1 September 1987.

NATIONAL RESEARCH COUNCIL, **The U.S. Machine Tool Industry and the Defense Industrial Base**, Washington, D.C., National Academy Press, 1983.

PALFRAMAN, Diane, "FMS: Too Much, Too Soon," **Manufacturing Engineering** (March 1987), 34-38.

PIORE, Michael J. and Charles F. SABEL, **The Second Industrial Divide: Possibilities for Prosperity**, New York, Basic Books, 1984.

PORTER, Michael E., **Competitive Strategy: Techniques for Analyzing**

Industries and Competitors, New York, The Free Press, 1980.

_____, Competitive Advantage: Creating and Sustaining Superior Performance, New York, The Free Press, 1985.

RAMAMOORTHY, C.V., et al., "Software Engineering: Problems and Perspectives," Computer (October 1984), 191-209.

SABEL, Charles, et al., "How to Keep Mature Industries Innovative," Technology Review (April 1987), 27-35.

SAKATA, Kazushi (Former Deputy General Manager, Hitachi Software Works), Interview 9/10/85.

SCHMENNER, Roger W., Production/Operations Management: Concepts and Situations, Chicago, Science Research Associates, 1984.

SCHONBERGER, Richard J., Japanese Manufacturing Techniques, New York, The Free Press, 1982.

SCOTT, W. Richard, Organizations: Rational, Natural, and Open Systems, Englewood Cliffs, NJ, Prentice-Hall, 1981.

SHIBATA, Kanji (Manager, Engineering Department, Hitachi Software Works), Interview 9/19/85.

SHIMODA, Hirotsugu, Sofutouea kojo (Software factories), Tokyo, Toyo Keizai Shimposha, 1986.

SHOOMAN, Martin, Software Engineering: Design, Reliability, and Management, New York, McGraw-Hill, 1983.

SKINNER, Wickham, Manufacturing: The Formidable Competitive Weapon, New York, John Wiley & Sons, 1985.

STUCKI, Leon G., and Harry D. WALKER (Boeing Computer Services), "Concepts and Prototypes of Argus," in HUNKE, H., ed., Software Engineering Environments, Amsterdam, North-Holland, 1981.

TABACHNICK, Barbara G., and Linda S. FIDELL, Using Multivariate Statistics, New York, Harper and Row, 1983.

TOYOSAKI, Kenshiro, Manager, Videotex Department, Software Development Division, Nippon Telephone & Telegraph, Interview 3 September 1987.

U.S. DEPARTMENT OF COMMERCE, A Competitive Assessment of the U.S. Software Industry, Washington, D.C., International Trade Administration, 1984.

UTTAL, Bro, "Japan's Persistent Software Gap," Fortune (15 October 1984), 151-160.

WILLIS, R.R. (Hughes Aircraft), "AIDES: Computer Aided Design of Software Systems II," in HUNKE, H., ed., **Software Engineering Environments**, Amsterdam, North-Holland, 1981.

WOODWARD, Joan, **Industrial Organization: Theory and Practice**, London, Oxford University Press, 1965.

----, ed., **Industrial Organization: Behavior and Control**, London, Oxford University Press, 1970.

YOSHIDA, Tadashi, Deputy Manager, Quality Assurance Dept., Software Division, Numazu Factory, Fujitsu, Ltd. Interviews, 9/24/85, 7/31/86, 9/7/87.

ZAVALA, A., "Research on Factors that Influence the Productivity of Software Development Workers," SRI International (June 1985).

ZELKOWITZ, Marvin et al., "Software Engineering Practices in the US and Japan," **Computer** (June 1984),