The Process Spectrum in Software Development:

An Exploratory Survey and Interpretation

by

Michael Cusumano

WP# 2611
was 2111-89

April 1989

THE PROCESS SPECTRUM IN SOFTWARE DEVELOPMENT: AN EXPLORATORY SURVEY AND INTERPRETATION

INTRODUCTION

A question treated in fields ranging from history, sociology, and organization studies to economics and management is why and how firms evolve beyond "craft" or job-shop modes of production to benefit from disciplined engineering and large-scale factory operations. Some researchers have maintained that mass-production engineering and factory systems emerged after industries "matured" -- when product standards appeared and companies began paying more attention to design and process innovations that improved manufacturing efficiency (Abernathy and Utterback). The de-skilling or routinization of work, high-levels of control over production tasks and process flows, divisions and specialization of labor, mechanization and automation, interchangeable parts, and standardized designs were some of the major innovations that have come to characterize large-scale factory systems in a variety of industries (Woodward, 1965; Chandler, 1977; Hounschell, 1984). In more generic terms, these characteristics reflect "bureaucratic" organizational structures and marketing strategies focusing on low costs and low prices, perhaps at the expense of reductions in product variety or functionality (Porter, 1980).

While some academic researchers, managers, workers, and other critics have found fault with factory systems and bureaucracies for the constraints they place on employee discretion and creativity, bureaucratic approaches have become extremely common in the industrialized world because they are efficient, given the problems that most organizations face and the limited solutions or technologies most have at their disposal (Miewald, 1970; Perrow, 1972; Jacoby,

1973). Without bureaucratic engineering and factory systems, few people would ride in cars or have access to a wide range of industrial products and services. On the other hand, too much standardization or rigidity in products and processes can lead to customer and employee dissatisfaction and decrease the ability of firms to meet competitive challenges such as changes in consumer desires and technology. The classic example of this was Ford's dramatic decline in market share during the mid-1920s, despite rising levels of productivity, when market demand and competitor offerings shifted to a greater variety of differentiated products (Abernathy and Wayne, 1974; Abernathy, 1978).

There are, in addition, interdependent organizational, technological, and strategic issues: A structured, standardized process may be inherently unsuitable for a dynamic environment and an unstandardized technology requiring complex development tasks. In fact, variability in individual productivity and quality, along with continual evolution in technology and market needs, have prompted debates over whether software development will always be more like an "art" or a "craft" rather than a technology suitable for the discipline of engineering processes and factory-like organizations (Brooks, 1975; Shooman, 1983; Hauptman, 1986). It is this issue -- what do software producers look like, in terms of basic organizational and process characteristics, and what should they look like -- that prompted this study. The research began with an exploratory survey that tested a simple hypothesis:

Hypothesis: Not all managers in software firms ignore standardization and control over tools and processes, or reusability of components -- concepts associated with disciplined engineering and factory production in other industries. Rather, firms probably fall into a spectrum, with some managers emphasizing factory-like concepts more than others.

This paper describes the rationale behind different types of organizations and processes, the survey of managers, and the results, as well as offers a framework for understanding how firms may segment their products and processes, as well as utilize other measures to improve efficiency while retaining enough flexibility to offer customized or unique products and adapt to changes in technology or the marketplace. The analysis of specific facilities, including performance measures, is reported elsewhere and part of continuing research.[1]

RATIONALE FOR A PROCESS SPECTRUM -- EVEN IN SOFTWARE

Software products fall into two broad categories: basic or system software, and applications. Basic software serves to control the primary functions of computer hardware, and includes operating systems, database management systems, telecommunications monitors, computer-language translators, and "utilities" such as program editors. Applications software sits, at least figuratively, on top of basic operating systems, and performs specific "user-oriented" tasks. These again include pre-written or "packaged" programs such as for payroll accounting, spreadsheet analysis, word processing, and other standard operations, as well as "custom" software written for specific tasks and customers, such as in the banking, financial services, manufacturing, government, or defense sectors. Another type of application are programs written as parts of integrated hardware and software systems (U.S. Department of Commerce, 1985).

One way to view the design and production of these programs is as a process of analysis and translation: analyzing a problem and then breaking it down into a series of smaller tasks expressed in manner ultimately

understandable to a computer. Programmers begin by translating problems into design specifications and then design specifications into "source code" written in a high-level (English-like) computer language. The next step is to translate the high-level program into a lower-level machine-language called "object code," consisting of zeros and ones that serve as instructions for the computer hardware. Special computer programs called compilers usually perform this transformation automatically, although design of the compilers, as well as prior steps in program development, frequently require considerable human thought and judgement (Arden: 564).

The development cycle continues in that software must still be tested and frequently changed, repaired, or enhanced (maintained), before and after delivery to users. In terms of time and costs, excluding those incurred during operations and maintenance, testing is usually the most labor intensive phase, followed by implementation (detailed design and coding) and high-level design. For a product that continues in service with periodic modifications, post-delivery maintenance may become by far the most costly activity, consuming as much as 70% of total expenditures over its lifetime, according to commonly-cited data on life-cycle costs (Boehm, 1976; Ramamoorthy et al., 1984).

While most software projects go through similar phases that appear sequential, the production process is more iterative, i.e. developers go back and forth among requirements analysis, specification, program design, coding, testing, redesign, re-testing, and so on. Experienced programmers or managers may give precise estimates of the time and labor each phase requires for particular applications, although numerous uncertainties upset schedules and budgets and thus make at least some software development something less than an exact science or engineering discipline. For example, project requirements might contain new functions that become difficult to build, customers often

change their minds about features they desire, and programmers usually take different amounts of time to perform similar operations. Furthermore, the larger projects become in terms of total length of code, numbers of components, and numbers of people, the more complex the interactions required become, and the greater the uncertainty of the final product's cost, schedule, and performance.

Variations in programmer performance appear to stem from differences not simply in native ability, but also in the particular experiences of the individual. A programmer who has written an inventory-control system in the past probably can complete another inventory-control system in less time than it would take a novice. The relationship of experience to productivity reflects the reality that software, though a generic type of product or technology, may vary enormously in content. The number and type of operations it must perform in each application greatly affect the amount of thought, time, and experience required to solve problems and write computer code. The need to adjust to each application or situation makes it difficult for producers to establish and maintain standards, controls, and schedules, as well as divide labor, automate tasks, and reuse components -- the essence of factory production.

Software thus appears to have characteristics -- little product or process standardization to support economies of scale in production operations, wide variations in project contents and work flows, planning and production tasks that are difficult to divide or de-skill -- that make disciplined engineering or factory-like operations difficult and perhaps unwise or impossible to introduce for many firms and applications. Software producers have to contend not only with constant evolution in technology and customer requirements, but also with demand for products with customized features.

To accommodate such a "non-routine" technology and dynamic environment,

many software producers embrace highly flexible, even "ad hoc" organizational structures and production processes, with little use of formal procedures or standards, such as in job shops or craft production in other industries. They hire mainly experienced or talented programmers, provide a loose set of work guidelines and product specifications, and rely on small teams to complete projects aimed either at customizing a product for a particular customer or developing a "packaged" program for mass-replication and distribution. These practices were essential in the early days of the industry, when product requirements were new and changing constantly, and programs were small, due to hardware limitations. Craft-oriented job-shop approaches continue to work well when product requirements are new or ill-defined, and software projects can be completed by a small group of people working in an integrated team.

But, for many companies and in many situations, while small teams of experts may be desirable, they are not practical. Shortages of skilled people, lengthy and complicated projects to finish within time or budget constraints, and products that different sets of people must maintain in the future, provide huge incentives for managers to adopt a more structured process -- covering methods, tools, procedures, controls, worker skills, and product components or designs -- to reduce skill requirements and systematically recycle process or design knowhow and other key factors of production among different projects. To structure software development or similar technologies successfully, however, managers must solve two fundamental problems that require linkages in competitive strategy and market positioning, organizational structure and personnel management, and product and process technology development: **when** to introduce a more formalized and standardized process; and **how** to achieve a balance between efficiency and flexibility acceptable to workers and customers as well as supportive of the firm's competitive positioning.

This very practical dilemma for managers reflects a long-standing academic debate. One set of positions, represented by various contingency theorists, range from the assertion that there is no one best way to do anything to the belief that optimal selections are contingent on if not determined by factors such as the stability of the environment (Lawrence and Lorsch, 1967), the characteristics of the technology (Woodward, 1965; Perrow, 1967), the size of the organization (Blau and Schoenherr, 1971), or political processes within the firm (March and Simon, 1958; Pfeffer, 1981). Others argue that managers can significantly shape the structure of their organizations and the technology--tools, techniques, information, and other elements used to make products or services -- they require by selecting different competitive positions (Child, 1972; Chandler, 1962 and 1977; Miles and Snow, 1978).

Contingency factors appear to account for no more than 50 to 60 percent of the variability in structure among organizations (Pugh, 1973; Robbins, 1987: 176). But if they exert even this much influence on the options open to managers, and optimal choices, then one might hypothesize that all software producers -- or all "successful" software producers -- should look like unstructured, highly flexible job shops, not "factories." But there should also be exceptions -- if managers can identify segments of their industry where relatively standardized processes and product components are appropriate at least for some customers, and then introduce a more structured or even "bureaucratic" system. This would require not simply an intermediate position between craft and factory production in terms of volume of output or scale of operations. Factory-like software facilities would have to reconcile seemingly contradictory elements, such as the ability to standardize process, skills, and perhaps product components, but still produce unique or customized products rather than engage in mass production. These organizations would also have to

evolve along with the technology, or risk falling hopelessly behind industry leaders and customer requirements.

The characteristics factory-like software facilities need to embrace also would make it difficult to categorize them in terms of conventional organizational designs or economic justifications based on economies of scale and mass production (Mansfield, 1985). An example is Joan Woodward's identification of three basic types of production: unit or craft production; mass production; and continuous processing, as in chemical manufacturing. In unit production, which relied on highly skilled workers, little formalization of organizational rules and procedures, and little centralization of decision-making authority, tasks were non-routine and usually manual. Mass production dealt with more complex but repetitious operations susceptible to standardization, economies of scale, formalization, divisions of labor, and mechanization. Continuous processing, since it was highly automated, did not usually require extensive centralization, formalization, or divisions of labor (Woodward: 35-50).

At first glance, software development, except for electronic replication of finished programs, appears to fall neatly into the realm of unit or craft production -- making one product at a time for a specific customer or set of requirements, rather than mass-producing components and finished products through a sequential assembly process or continuous automated operation. Yet, some software producers began adopting formal rules and procedures, and divisions of labor, as early as the 1960s. They also managed to "mechanize" some operations, i.e. support them through computer-aided systems, and "automate" others, i.e design them to be performed automatically, with little or no human intervention. Thus the histories of actual firms suggests that not all software development falls into the category of unit or craft production, not even in the early days of the industry.

A classification problem arises with another widely-accepted organizational scheme elaborated on by Henry Mintzberg (Mintzberg, 1979). In this terminology, factory-like software facilities probably came closest to a "professional bureaucracy" -- with standardized but specialized skills, procedures, and tools controlling the work process rather than a conventional administrative apparatus. The notion that professionalization of the work force can serve the same functions as bureaucratic administration in mass-production industries, while allowing for more flexibility to adapt to different customer needs or changes in the work flow, dates back to the work of Arthur Stinchcombe on the construction industry in the late 1950s (Stinchcombe, 1959). Other examples of professional bureaucracies include engineering consultants and hospitals.

But descriptions of professional bureaucracies claim they rely on decentralized structures -- lots of individual or small-group activities -- and little formal divisions of labor. In contrast, accounts of factory-like approaches to software development in Japan (Matsumoto, 1987; Cusumano, 1989) as well as in the U.S. (Orlikowski, 1988), sometimes located teams at customer sites, although they generally did this in the planning and high-level design phases and attempted to centralize detailed design and programming operations in large-scale, capital-intensive facilities. Professional bureaucracies, by definition, also consisted of professionals that had years of education in specific fields. Software producers nominally adopting factory practices hired mainly college graduates unskilled in software and then trained them to use a standardized methodology and tool set (built, of course, by more expert people).

In this sense, at least some software facilities contained elements characterizing mass-production factories in other industries, which some theorists have labeled "machine bureaucracies" (including divisional structures in

large organizations): standardized tasks, unskilled employees grouped into functional departments, centralized authority, high levels of rigid mechanization or automation, and formal divisions of labor. Yet tasks in factory-like software facilities did not seem completely de-skilled; most projects required technical abilities, adaptation, and thought on the part of designers and people who built and tested products. Functional departments and divisions of labor also did not appear so rigid, and managers routinely used matrices -- project teams consisting of members borrowed temporarily from functional departments-- which Mintzberg and many others have associated with a loosely structured "adhocracy," a term referring primarily to teams of specialists relying on ad hoc structures and procedures (Mintzberg, 1979: 431-467).

These organizational types contrast to a "simple structure," characterized by little or no formal procedures or processes, decision making done by one person or a small group, such as in a small family-owned store or entrepreneurial firm. Simple structures were apparently common in the early days of the software industry and still seem to characterize small "software houses." Yet, as many software producers have discovered, once organizations and production operations grow over time, more elaborate structures and controls become essential (Woodward, 1965; Mintzberg, 1979).

Another classification scheme, suggested by Charles Perrow, is more useful in that it focuses on task variability and how personnel analyze problems (Table 1). Organizations dealing with "routine technologies" encounter few exceptions and thus face problems that, over time, become easier to analyze and solve through formal procedures or tools. This standardized process eliminates the need to have large numbers of highly-skilled (and usually expensive) employees capable of re-inventing solutions each time problems occur. Firms dealing with "engineering technologies" have more exceptions, but these are still relatively

well-defined and, according to Perrow, can be managed systematically. Perrow contrasts routine and engineering technologies with "craft" technologies, defined by a limited range of variability but problems that were ill-defined and difficult to analyze, as well as "non-routine" technologies, which indicated many exceptional and difficult tasks (Perrow, 1967).

But even this scheme does not adequately categorize factory-like software organizations, which clearly exhibited some features of routine, mass production. Factory-like software facilities appeared to standardize only **some** tasks that were only **relatively** de-skilled; they relied on only **some** divisions of labor and only **relatively** formalized rules and procedures -- compared to modes of operations before managers explicitly adopted more structured approaches, or to explicitly "adhocratic" firms or organizations with little or no formal structures.

THE SURVEY OF MANAGER EMPHASES

The claims of some software producers to have adopted factory-like practices, in both Japan and the U.S., prompted this author to identify and examine those companies that explicitly tried to organize the production of commercial software (programs written for sale or inclusion with hardware) by establishing what managers called "software factories" or, at least, by adopting what managers termed factory strategies and objectives. These firms consisted of System Development Corporation (SDC, currently a division of Unisys) in the U.S., which launched a small factory in the mid-1970s before disbanding it after three years due to the preference of project managers to development software in integrated groups at customer sites in the U.S. (Cusumano, 1988a); and Hitachi, Toshiba, NEC, and Fujitsu in Japan (Cusumano 1987a, 1987b, 1987c, 1988b), which launched several factories or factory-like efforts between 1969 and the early 1980s (Table 2). While in certain key respects the customers SDC

aimed at were very different from those the Japanese served, these firms were comparable to the extent they all made unique systems or customized applications software for large computers, and had to manage lengthy and often extremely complex projects.[2]

A problem with this sample arose in that, while the term "factory" was particularly popular or acceptable in Japan, U.S. and European firms after SDC tended not to use the term factory except to describe standardized tool and methodology sets. But there were estimates of as many as 200 enterprises in the U.S. alone with more than 1000 software personnel in centralized facilities, emphasizing -- at least to some degree -- standardized designs and reusable code components, common tool development, formal testing and quality assurance procedures, productivity measurement and improvement efforts, and process research (Jones, 1986: 243). In addition, interviews and historical research indicated that IBM in the U.S. was probably the first company in the world to create a structured, bureaucratic process and organization for software, which it did in the mid-1960s to develop operating systems for the System/360 family of computers. Yet IBM has never used the term "software factory," and has continued to label its facilities "laboratories" or "programming centers." Thus the existence of large software producers around the world, some explicitly using the term factory and others avoiding it though seeming to follow some factory-like practices, indicates that mere adoption of a name, though it may reflect specific management objectives, is not in itself meaningful.

To provide some perspective on what managers emphasized apart from the labels they used, a next step in the research was to survey managers at 52 large software facilities (25 Japanese, 26 U.S., 1 Canadian) at 30 companies in North America and Japan. The survey was exploratory in the sense that it

simply tested to see if current managers embraced emphases adopted earlier in the SDC Software Factory, rather than trying to present a definitive model of what constituted a "factory" approach.

The published descriptions and stated objectives for the SDC Software Factory provided a comprehensive formulation of what a basic factory for software might look like. In particular, these materials suggested eight criteria relating to inputs standardization (emphasis on reuse of software code) and tool or process standardization and control. SDC relied on or hoped to build 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 to guide individuals and groups constructing different parts of a program; and an on-line computerized interface linking various tools and databases. These five variables constituted the core process and tool questions in the survey. Since another type of factory strategy should be to produce standardized components and then to reuse them, rather than "reinventing the wheel" with every customer order, three questions were included about design for reuse, execution of reusability, and control (monitoring of reuse rates).

Major software producers in Japan and North America were identified through literature surveys and lists of software producers; further investigation led to the identification of senior managers either responsible for overall software engineering management or with responsibilities over several projects and with sufficient experience to present an overview of practices for an entire facility or product division. The intention was to study manager emphases at the facility or product-division 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.

Managers who agreed to participate in the survey received a questionnaire containing the eight core questions plus more than a dozen others asking for supplementary data.[3] For the core questions, they had to rank their emphasis and impression of general policy at their facilities on a scale of 0 to 4, as well as to comment on each answer. Optional questions also requested performance measures such as actual rates of reused code in a recent sample year. The intent of the survey and meaning of questions was explained at least to the individuals in each firm handling distribution of the questionnaires. Japanese managers were sent questionnaires in English but asked to comment on each question either in Japanese or English.

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). The analysis that follows further broke these down into telecommunications software, 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 the other firms contacted completed the survey. To check for consistency in answers, two managers at each firm or facility were asked to respond, although only about one-third the companies returned two completed surveys for each type of facility. Among those, the answers were similar, differing by only a few percentage points, and therefore were averaged. Two thirds of the answers,

however, represent single responses.[4]

RESULTS

A factor analysis procedure with varimax rotation indicated that the eight questions constituted two approximately orthogonal factors, listed as the inputs and tool and process dimensions in Table 3. Both factors had an eigenvalue rounding to approximately 1.0 or higher and together explained nearly 82% of the variance in the survey answers; the inputs dimension alone accounted for 62% of the variance. For each dimension, the variables with a strong loading (minimum 0.4) were summed and used to test differences in the average Japanese and North American scores, as well as to test if product type or country of origin of the facility were significantly correlated with the process and reuse scores.[5]

The data reported in Table 4 reflects scores for each dimension; Table 5 summarizes the average Japanese and North American responses to the inputs and tools/process dimensions. Table 6 presents the results of analysis of variance tests to determine the effects of product types or country of origin on the scores reported for the two dimensions. Tables 7 and 8 compare reuse rates reported by the Japanese and North American facilities, and analyze correlations with type of product and country of origin. Table 9 is a regression analysis looking at the correlation between manager responses and reported reuse rates.

The results support the hypothesis that there is a spectrum among managers in how they answered the survey questions. These answers also corresponded closely to case studies in progress of individual firms. Thus it clearly seems that, 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 were able to place more emphasis on control and standardization of inputs (reusable modules of code) as well as basic tools and process questions. The analysis of variance tests confirmed that product types, at least defined generally, had no significant impact on where managers scored on either of the dimensions surveyed.

The data also confirm there are probably national differences in reusability emphasis. Japanese firms scored much higher on the inputs (8.7 to 5.9) dimension (significant at 0.001), while there was no significant difference in Japanese and North American responses on the tools and process dimension. Reported actual reuse rates in Japan were also significantly higher than in North America (34.8% versus 15.4%), across all product types. The reuse data are very tentative and subject to different methods of counting across firms. Nonetheless, they suggest that Japanese applications producers, who clearly are marketing customized products, as well as Japanese systems producers, who sell basic software, both tend to rely on reused code. Recycling standardized components at Japanese applications producers is probably due to the huge demand for custom software, as opposed to packages (low demand for which appears to stem from the desire of Japanese firms to have unique features in their software), and the expense of writing similar applications for different customers from scratch with each order (Table 10).

The emphasis on reuse in commercial operating systems and other types of software also seemed to reflect process decisions to maximize efficiency, even with nominally unique products. Compatibility of a firm's hardware architectures and operating systems across different size machines, such as with Digital's VAX line, is an important factor facilitating reuse. As indicated in Table 9, however, it also appeared that manager emphases had some impact of this variable. There was a significant correlation between high emphases on

reusability and high reported rates of actual reuse.

Figure 1 presents another configuration of the survey responses. Some clearly fell into the upper right-hand corner of the matrix and thus could be characterized as "job shops," with little or no emphasis on standardization, control, or reuse. Those in the lower left appeared more like "flexible" factories or integrated design and production systems in the sense that managers strongly emphasized reuse as well as standardization and control in the design and construction of new products that were both unique and customized for different customers. Most responses also fell on the right side of the matrix, and U.S. responses on tool and process questions were not significantly different from the Japanese, though most (but not all) of the facilities in the lower right were Japanese. The survey of managers thus indicated that (1) a spectrum, including factory-like emphases, existed in both Japan and North America, independent of product type; and (2) Japanese firms were significantly different at least in their emphasis on one factory-like characteristic: reusability.

Some firms in the survey appeared to be in the "high-end" of the market. These included Draper Laboratories, Honeywell, and Nippon Electronics Development, which designed unique command-control missile systems, satellite-control systems, and other programs largely for government or specialized use. They did not emphasize reuse of code across different projects possibly because most of their work was unique from job to job. However, managers at some of their direct competitors -- Unisys/SDC, TRW, Unisys/Sperry, Toshiba--placed much more emphasis on reuse as well as other measures of process control or tool standardization and integration that might be associated with a more factory-like approach. This again suggests that various process approaches are possible even in similar market segments.

The only firms in this sample that developed applications packages on a significant scale were IBM, Cullinet, and Computervision. They also showed a range in emphases, suggesting IBM placed relatively more emphasis on reuse, while Computervision appeared to operate more in a job shop or perhaps laboratory mode, with very little emphasis on the variables in the survey.

There are several caveats to this survey, however. . It, of course, represents no more than a sampling of the self-reported opinions of one or two middle managers from major software producing firms, rather than a comprehensive analysis of actual practices in projects done within product groups in particular firms. Managers might be exaggerating or understating their emphases on the various questions, although the respondents were carefully selected and an attempt was made to examine the comments and other documents, such as technical articles, and to interview managers in person or by phone, and visit some actual sites, to see if answers corresponded to realities. Detailed interviews and/or site visits were conducted for NT&T, Mitsubishi, Fujitsu, NEC, Hitachi, Digital, IBM, Data General, Unisys/SDC, Draper, Nippon Systemware, Hitachi, Hitachi Software Engineering, and Nippon Business Consultants.

Another reservation regarding the survey is that, although managers were asked to report on general practices in their areas, some companies reported high levels of variability within projects. This appeared especially true in the cases of Digital and Hitachi Software Engineering. In Digital, while there were rigorous corporate guidelines, top management placed more emphasis on the characteristics of final products rather than the development process, and individual groups were allowed considerable autonomy, especially in applications. Thus, while there was a management policy of stressing reusability, some project managers, even within the VAX and VMS product areas, did not appear

to emphasize reuse at all.[6] In the case of Hitachi Software Engineering, some groups worked directly within Hitachi's software factories, following the factory procedures and using the factory tools with the same degree of conformity as Hitachi employees. Other groups worked on independent projects where customers determined the standards. For this reason, there was a large variation within Hitachi Software Engineering as a company and within product areas, and thus managers felt compelled to score themselves on the low end of the spectrum, even though they managed many projects using procedures identical to Hitachi's.[7]

Furthermore, since the sample size is relatively small in absolute numbers, the results of this analysis must be considered as no more than suggestive of managerial emphases existing at the participating software facilities and in the two geographic settings. It should be noted, however, that the surveyed Japanese firms accounted for what seems to be the majority of software commercially written and sold in Japan, while the surveyed North American firms included most of the large producers of operating systems and applications software, and other basic software products such as data bases.[8]

There is also other evidence supporting the observation that firms, in both software and computer hardware, position themselves through different combinations of price and product performance. Surveys of nearly 20,000 users of products offered by U.S. and Japanese vendors in the Japanese market indicated large variations in customer satisfaction, with Japanese firms trailing U.S. vendors mainly in basic software and Japanese vendors leading U.S. firms in areas such as hardware price-performance and applications system engineering. Furthermore, in examining pricing data, the same surveys indicated a clear spectrum among U.S. and Japanese vendors in this measure as well, with Japanese prices averaging about half those of U.S. vendors (Cusumano, 1988c).

INTERPRETATION: STRATEGIC POSITIONING

Compared to conventional production-system archetypes, factory-like software facilities in Japan or elsewhere seemed to fall somewhere in the middle of a continuum of production approaches and perhaps product types as well. In other industries, this spectrum stretched from loosely-structured job or craft shops on one end to highly structured organizations on the other, integrating engineering and factory operations oriented toward mass production (Figure 2). In software, case studies suggested that factory-like approaches, especially in applications areas, relied on tools and methods that made them resemble what might be termed "flexible design and production systems"' such as in application-specific semiconductor design and fabrication (ASIC), or integrated computer-aided design and manufacturing (CAD/CAM) and flexible manufacturing systems (FMS) in a variety of industries (Harvard Business School, 1986).

Figure 2 suggests that, in firms competing in various types of industries, this combination of flexibility and efficiency comes through the use not of fully-standardized methods, tools, components, and designs, but through limitations on the range of products and customers served, and planning for economies of scope -- savings from some joint use of inputs or factors of production that, once acquired to make a particular product, might be used to make others at less cost than would otherwise be the case, as in reuse of designs or process R&D (Baumol et al., 1982; Lorange et al., 1986). Also useful to achieve this balance are process and quality standardization and control, a combination of tailored and centralized process R&D, standardization of worker skills, some divisions of labor, systematic reuse of product designs or code, and extensive use of computer-aided tools. In practice, software firms did not use factory approaches for all types of products or customers, but only for those

that were relatively well-understood and had some commonality across different projects.

This notion of putting only some work in factories and less routine work in other facilities (less-structured subsidiaries, software houses, or laboratories, for example) reflects another observation about how managers in factory-like software facilities managed product development: They adopted different strategies and structures to compete in different segments of a larger market--software. Figure 3 summarizes how firms appeared to segment products, processes, and customers, based on a series of case studies. These suggest strongly that successful factory-like approaches targeted the "middle" of a market in terms of product price and functionality, stressing reusability and economies of scope in designs, methods, tools, and application-specific knowledge, and appealing to customers that seemed sensitive to a combination of price and product performance.

On the other hand, firms in both the high-end, "full custom" business and those making standardized packages appeared to require somewhat loosely-structured, project-centered approaches that were different with each product and relied heavily on personnel who were highly skilled and knowledgeable about particular applications. Unique projects and highly-skilled personnel were probably more expensive to manage, although this probably does not matter to the producer if customers of premium products or services pay adequately high prices or if a package becomes a "best seller" and reaps large revenues, since there should be minimal incremental expenses in software (excluding efforts needed for distribution and marketing), since replication of a program is a simple, nearly instantaneous electronic process.

Factory-like software facilities thus combined some of the flexibility of job shops, in that they made unique or customized software, with some of the

efficiency of factory modes of production. This positioning very much resembled the category of the "analyzer," posed by Miles and Snow. These firms adopted follower strategies in the market place, often allowing small, entrepreneurial firms ("prospectors") to lead in inventions. The analyzers then analyze these products and, through a more structured organization or process, try to introduce similar or superior products at lower prices. Yet, in combination with more controls and emphases on efficiency than the innovating firms, analyzers had to be flexible enough to respond quickly to the arrival of new products and markets. They also stood in contrast to "defenders," who attempt to hold high-priced niches or defend mass-market positions through cost reduction, and "reactors," who seem to follow no one particular strategy but mainly respond to the actions of others (Miles and Snow, 1978; Miles et al., 1978).

The electronic nature of production operations in software facilities brings up another distinctive characteristic of software factories that is not the focus of this study but which has become a major area of research in itself: Software factories represented a new type of production organization, where the "workers" were different from employees in conventional manufacturing. Not only was there was no conventional mass production; but even coding jobs were relatively skilled and technical, while most "work" consisted of design, testing, and redesign on computer screens, as well as meetings to coordinate divided and cooperative tasks, and filling in reports or writing documentation (Hirshhorn, 1984; Zuboff, 1986; Orlikowski, 1988).

It follows that managerial tasks in factory-like software facilities were also different. Conventional decision-making algorithms relying on economies of scale and learning curves to determine precisely the time and cost of different tasks did not readily apply. In fact, some writers even lamented the appearance

of "dis-economies of scale": average productivity levels decreasing as the number of members in a project, or the size of a program, rose beyond a certain manageable level (Brooks, 1975; Boehm, 1981; Banker and Kemerer, 1988). The reality of software development was that managers faced a complex, iterative series of design, product-construction, testing, and redesign operations, for products that were unique or customized and thus likely to encounter some unpredictable or new requirements. What software factories achieved was to place some boundaries on this unpredictability, in both products and processes.

CONCLUSION

It seems clear that software producers, like firms in other industries, do not all fit neatly into any one categorization. They were not conventional production organizations, which provided the impetus for categories such as job shop and factory. Still, firms seemed to fall into a spectrum -- in terms of manager emphases, prices, and customer responses to their products and services.

A related conclusion follows: The quest for an absolute answer to whether software development is or should be managed more like an art or craft rather than science or engineering is probably fruitless. This is because the nature of software development, and the optimal process or organization, depend on the specific tasks at hand. To the extent that these tasks differ with market segments, product types, and a firm's competitive positioning, whether software is appropriate for a factory-like process is a strategic choice subject to management discretion. In other words, management choices, in response to the characteristics of software products and customers, the technology, and perhaps features specific to programmers and programming organizations, can result in a range of process and structural variations in software organizations. The most

relevant concern for managers should thus be not how to label software development but how to create an effective balance: of efficiency, such as standards and tools; and flexibility -- the ability to adapt to different customer needs as well as change and evolve, through the use of effective but versatile procedures, methods, tools, controls, and components, as well as individual or organizational skills.

This study thus supports both strategic and contingency perspectives: Factory-like approaches to software development, though important for what they reveal about organizations and technology management, seem appropriate only for some product or market segments and some competitive strategies. Be that as it may, another thought emerging from this research may find little support among those who would insist software development forever remain an art or craft: Where more structured approaches appear possible and advantageous to introduce, adhocratic or job-shop practices represent a waste of human and capital resources, and an opportunity for management to improve the competitive capabilities of the firm.

There seem, in fact, many advantages to pursuing a more structured process, if at all possible. Job- or craft-oriented approaches make it difficult for software developers to share experiences in problem solving and apply potentially useful solutions arrived at in one project -- such as tools, methods, procedures, and product designs -- to other projects. One might even argue that both the difficulty or inappropriateness of product and process standardization across different projects, as well as insufficient efforts toward this end, contribute to the recurrence of similar problems year after year. Solving the same problems over and over again wastes valuable human effort, taking away time that skilled personnel might spend in improving products or inventing new ones or new processes. The task facing software producers -- as

many of their counterparts in other industries have already realized -- is not only to remain adaptable to change but to identify areas of repetition or recurring problems and then create a strategy and infrastructure -- of tools, methods, reusable designs, and people -- that might turn this more systematic management of product and process development into a competitive variable.

Table 1: SOFTWARE-FACTORY CASE STUDIES

Key: BS = Operating Systems, Database Management Systems, Language Utilities, and Related Basic Software
App = General Business Applications
RT = Industrial Real-Time Control Applications
Tel = Telecommunications Software (Switching, Transmission)

Notes: All facilities develop software for mainframes or minicomputers.
Products and Employee figures refer to 1988 estimates, except for SDC.

Est.	Company	Facility/Project	Products	Employees
1969	Hitachi	Hitachi Software Works	BS	1500
1975-76	SDC	Santa Monica Software Factory	RT	200 (1976)
1976	NEC	Software Strategy Project		
		(Fuchu)	BS	2500
		(Mita)	RT	2500
		(Mita)	App	1250
		(Abiko)	Tel	1500
		(Tamagawa)	Tel	1500
1977	Toshiba	Fuchu Software Factory	RT	2300
1979	Fujitsu	Kamata Software Factory	App	1500
1983	Fujitsu	Numazu Software Division (Numazu Works est. 1974)	BS	3000
1985	Hitachi	Omori Software Works	App	1500

Source: Cusumano, 1989: 26, based on company data and interviews.

Table 2: ORGANIZATIONAL STRUCTURE AND TECHNOLOGY

Structure	Technology	Tasks & Problems	Characteristics
Machine Bureaucracy	Routine, Mass Production	Few exceptions, well-defined	Standardized and de-skilled work, centralization, divisions of labor, high formalization of rules and procedures
Professional Bureaucracy	Engineering	Many exceptions, well-defined	Standardized and specialized skills, decentralization, low formalization
Adhocracy	Non-routine	Many exceptions, ill-defined	Specialized skills but few or no organization standards, decentralization, low formalization
Simple Structure	Unit or Craft	Few exceptions, ill-defined	Few standardized specialized skills, centralized authority but low formalization

Sources: Woodward, 1965; Mintzberg, 1979; Perrow, 1967 and 1972; Robbins, 1987.

Table 3: SURVEY AND SAMPLE OUTLINE

SAMPLE: N = 51 (25 Japanese, 26 U.S., 1 Canadian)
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

SURVEY QUESTIONS:

Dimension I: Inputs Standardization (Max. Score = 12)

1. 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

2. 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.

3. Monitoring of how much code is being reused

Dimension II: Tool and Process Standardization and Control (Max. Score = 20)

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.

6. A centralized program library system to store modules and documentation.

7. 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.

8. 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.

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

Table 4: SUMMARY AND RANKING OF SURVEY SCORES (%)

SAMPLE: N = 51 (25 Japanese, 25 U.S., 1 Canadian)

Notes: * Indicates Japanese facilities
@ Indicates averaged responses

	Max. Score =	12	20
COMPANY/FACILITY		**Inputs**	**Tools/Process**
Telecommunications Software			
*NT&T Applications@		11	16
*Mitsubishi Electric		9	17
*Fujitsu Communications		9	15
*NEC Switching Systems		9	14
AT&T Bell Labs Applications)		7	16
Bell Communications Research		6	15
*Hitachi Totsuka Works		6	13
*NT&T Systems@		6	12
Bell North Research		5	20
Commercial Operating Systems			
Digital VAX (Layered Products)		11	16
*NEC Software, Ltd.		10	18
*NEC Fuchu Factory		9	18
IBM-Endicott		8	17
*Hitachi Software Works@		8	15
Control Data@		7	17.5
Digital VMS		7	16
*Fujitsu Numazu Factory@		7	16
*Mitsubishi Electric		7	12
Unisys/Sperry@		5.5	12
Data General		5	13.5
IBM-Raleigh		2	17
Real-Time Control Applications			
*Toshiba Software Factory		12	16
*NEC Industrial Systems		12	16
Unisys/SDC		11	14
*Hitachi Omika Works		10	15
TRW		9	20
Unisys/Sperry@		8	20
*Mitsubishi Electric		8	15
Hughes Aircraft		5.5	17
Boeing@		3	16
Honeywell@		2	10
Draper Laboratories@		1	6.5

Table 4 continued

	Max. Score =	12	20
COMPANY/FACILITY		**Inputs**	**Tools/Process**
Industrial Operating Systems			
*Toshiba Software Factory		12	16
Boeing@		3	15
Business Applications			
*Nippon Systemware		11	14
*Nippon Business Consultants		10	11
*Fujitsu Kamata Software Factory@		9.5	15
Martin Marietta/MD		9.5	13
*NEC Information Services		9	19
Control Data@		9	18
*Hitachi Omori Works		8	15
*NEC Mita		8	15
*Hitachi Software Engineering@		7.5	9
IBM (Office Products)		7	16
Arthur Anderson		7	20
EDS/GM@		6	12.5
Cullinet		6	13
*Nippon Electronics Development		6	7
*Mitsubishi Electric		4	10
Martin Marietta/Denver		3	14
Computervision@		3	6
Digital (Educational Products)		3	5

Table 5: COMPARISON OF AVERAGE JAPANESE AND N.A. SURVEY SCORES

Note: Standard Deviations are in parentheses

	n = 25	n = 27	N = 52
Dimension	**Japanese**	**N.A.**	**Sample Average**
Inputs	8.7 (2.1)*	5.9 (2.7)	7.3 (2.4)
Tools/Process	14.4 (2.9)	15.0 (3.7)	14.7 (3.3)

* $p < 0.01$

Table 6: EFFECTS OF COUNTRY AND PRODUCT TYPE

Test: ANALYSIS OF VARIANCE
N = 52 (25 Japanese, 26 U.S., 1 Canadian)

Effects on INPUTS Score:

Variable	F-ratio	Sig. Level	Degrees of Freedom
Country#	17.128	.0002	1
Product Type##	.200	.9367	4

Effects on TOOLS/PROCESS Score:

Variable	F-ratio	Sig. Level	
Country#	.066	.8013	1
Product Type##	1.395	.2522	4

Coded as 0 = Japanese facility, 1 = North American facility
Coded as 1 = Telecommunications Software, 2 = Commercial Operating Systems, 3 = Industrial Operating Systems, 4 = Real-Time Control Applications, 5 = General Applications

Table 7: COMPARISON OF REPORTED JAPANESE AND N.A. REUSE RATES

n = 18	n = 18	N = 36
Japanese (Std. D.)	**N.A. (Std. D)**	**Sample Average**
34.8% (18.3)*	15.4 (14.1)	25.1 (16.3)

$p < 0.01$

Table 8: EFFECTS OF COUNTRY AND PRODUCT TYPE ON REUSE RATES

Test: ANALYSIS OF VARIANCE N = 36

Variable	F-ratio	Sig. Level	Degrees of Freedom
Country#	12.728	.0014	1
Product Type##	1.188	.3395	4

Coded as 0 = Japanese facility, 1 = N.A. facility

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

Table 9: REUSE EMPHASIS AND REPORTED REUSE RATES

Test: MULTIPLE REGRESSION

36 observations fitted, forecast(s) computed for 15 missing values of dependent variable

Ind. Variable	Coeff.	Std. Error	t-value	Sig. Level
Constant	3.234807	14.155589	0.2285	0.8207
Inputs	3.295832	1.036561	3.1796	0.0033
Tools	-1.080677	1.47857	-0.7309	0.4702
Process	0.450292	1.588475	0.2835	0.7786

R-SQ. (ADJ.) = 0.1876
SE= 16.996120
MAE= 12.130720
DurbWat= 2.020

Table 10: JAPAN-U.S. HARDWARE AND SOFTWARE COMPARISON, 1987

Notes: Japanese Yen converted at $1.00 = 125 Yen
NA = Not Available
Custom Software/System Integration for Japan includes consulting ($.67 billion); for the U.S. market, this category refers to contract programming and design

	Japan		U.S.	
Total Market	$34.1		$70.4	
Software Revenues/Total Market		38%		35%
Hardware Shipments	$21.0	100%	$45.6	100%
Large Systems	8.7	41	9.1	20
Medium Systems	3.1	15	8.7	19
Small Systems	5.0	24	8.2	18
Personal Computers	4.2	20	19.6	43
Software-Vendor Revenues	$13.0	100%	$24.8	100%
Total Packages	1.4	11	13.1	53
Types:				
(Systems/Utilities)	NA	--	(5.0)	(20)
(Application Tools)	NA	--	(3.7)	(15)
(Application Packages)	NA	--	(4.5)	(18)
Custom Software/System Integration	10.1	78	9.6	39
(Custom Software Only)	(7.9)	(61)	NA	--
Facilities Management/Maintenance	1.4	11	2.1	8
Miscellaneous Data:				
1987-1992 Compound Annual Growth Estimate for Software Revenues		17%		20%
Annual Growth in Supply of Programmers		13%		4%
Typical Wait for Customized Programs in Months (ca. 1984)		26		40
Computer Makers as Suppliers:				
of Basic Systems Software		70%		45%
of Applications Software		15%		5%

Sources: International Data Corporation, "Japan Computer Industry: Review and Forecast, 1987-1992," January 1989; and International Data Corporation, Computer Industry Report: The Gray Sheet, 16 December 1988, p. 3. Also, for miscellaneous data, U.S. Department of Commerce, A Competitive Assessment of the U.S. Software

Industry; A. Zavala, "Research on Factors that Influence the Productivity of Software Development Workers," Palo Alto, Calif., SRI International, June 1985; H. Aiso, "Overview of Japanese National Projects in Information Technology," International Symposium on Computer Architecture, Lecture 1, June 1986, Tokyo; Fumihiko Kamijo, "Information Technology Activities in the Japanese Software Industry," **Oxford Surveys in Information Technology**, Vol. 3, 1986.

Figure 1: EMPHASIS ON REUSE VERSUS TOOL AND PROCESS STANDARDIZATION

N = 51 (25 JAPAN, 25 U.S., 1 CANADA)

CODES: 0 = JAPANESE FACILITIES

1 = U.S. AND CANADIAN FACILITIES

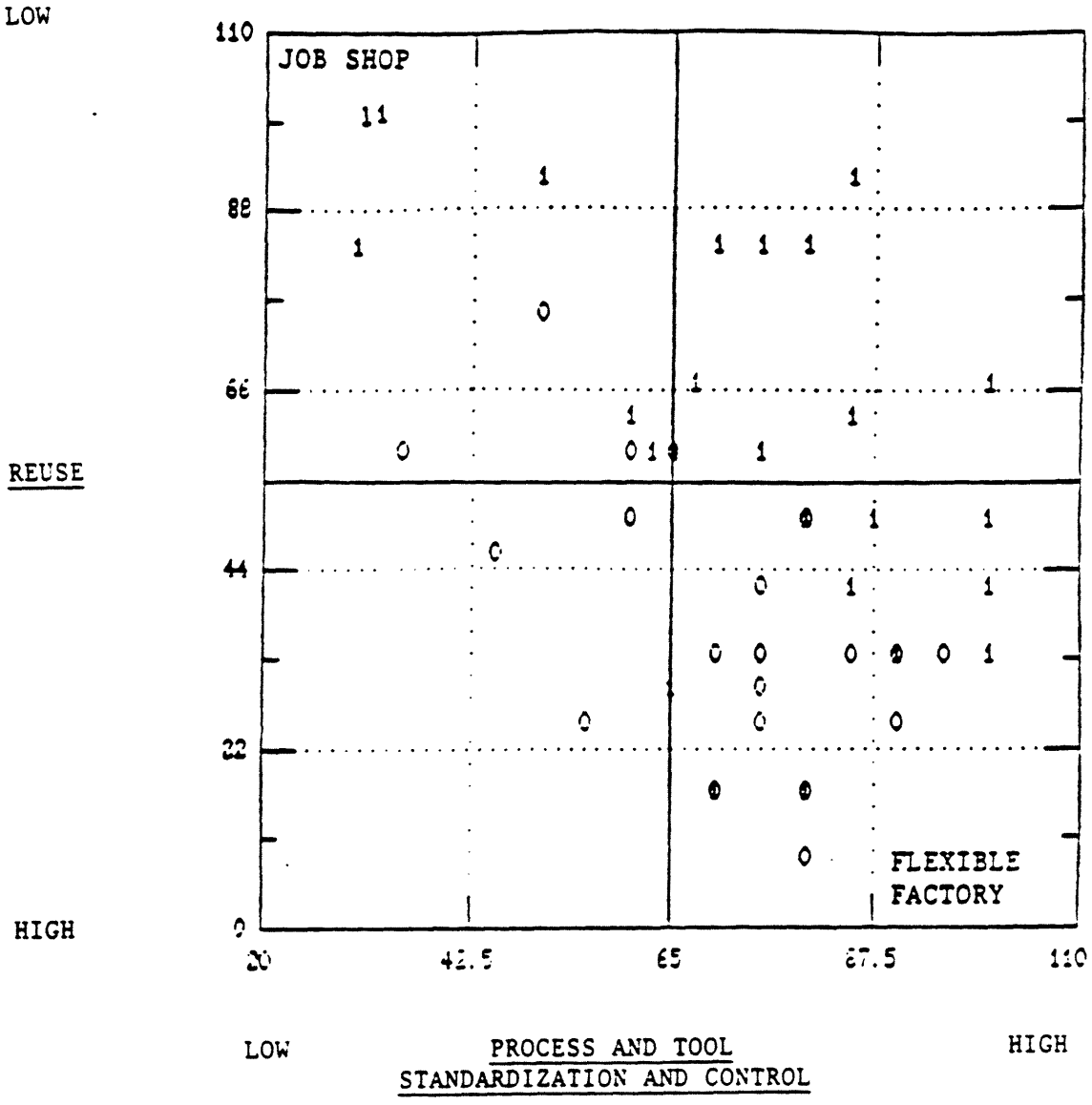

Figure 2: SPECTRUM OF PRODUCTION APPROACHES

CRAFT PRODUCTION OR JOB-SHOP

Strategy: Customize Products and Processes for Individual Customers

Implementation:
- Little Strategic Integration Beyond the Individual Project
- Nearly Unlimited Range of Products and Customers
- Few Economies of Scale or Scope
- Focus on Product and Process Flexibility
- Little Process and Quality Standardization or Control
- Project-Centered Process R&D
- Dependence on Highly Skilled, Multi-Functional Workers
- Little Functional Divisions of Labor
- No Systematic Reuse of Product Components
- Little Capital-Intensive Automation

Tradeoff: Product-Process Flexibility Over Process Efficiency

---

FLEXIBLE DESIGN AND PRODUCTION

Strategy: Efficient Production of Different Products

Implementation:
- More Strategic Integration and Management
- Broad But More Limited Range of Products and Customers
- Planned Economies of Scope More Than of Scale
- Focus on Process Analysis and Improvement
- More Process and Quality Standardization and Control
- Tailored and Centralized Process R&D
- Standardization of Worker Skills
- Some Functional Divisions of Labor
- Some Systematic Reuse of Product Components
- Extensive Use of Computer-Aided Tools

Tradeoff: Effort and Risks Required to Balance Efficiency and Flexibility

---

MASS-PRODUCTION ENGINEERING AND MANUFACTURING

Strategy: Mass Production of Standardized Products

Implementation:
- High Level of Strategic Integration and Management
- Narrow Range of Products and Customers
- High Economies of Scale
- Focus on Process Standardization and Efficiency
- High Process and Quality Standardization and Control
- Tailored and Centralized Process R&D
- Highly Standardized Worker Tasks and Skills
- Rigid Functional Divisions of Labor
- Reuse of Interchangeable Product Components
- Rigid Automation

Tradeoff: Process Efficiency Over Product-Process Flexibility

Figure 3: TYPOLOGY OF PRODUCT-PROCESS OPTIONS

Market Variety	Product-Process Type	Implementation & Organization Design	Production	Strategy:
Infinite	Full Custom	**Software or Conventional Products:** **CRAFT PRODUCTION OR JOB SHOPS**		Product Performance and Process Flexibility
		Product & System Engineering	Batch Production	Customer Premiums
Medium	Systematic Reuse	**Software or Conventional Products:** **FLEXIBLE DESIGN AND PRODUCTION**		Economies of Scope: Inputs, Process
		Product Engineering	Product Construction	Customers Discriminate on Price and Product Features
		CAD/CAM, FMS Program Generators		
Few	Full Standard	**Conventional Products:** **MASS-PRODUCTION ENGINEERING & FACTORY SYSTEM**		Low-Cost, Standardized Products Economies of Scale & Mass Production
		Software: **PROJECT, LABORATORY** (packages)		Low Margins but High Unit Sales

REFERENCES

Abernathy, William J.
1978 The Productivity Dilemma: Roadblock to Innovation in the Automobile Industry. Baltimore, Johns Hopkins University Press.

Abernathy, William J., and James Utterback
1982 "Patterns of Industrial Innovation." In Michael L. Tushman and William L. Moore, Readings in the Management of Innovation, 97-108. New York, Pitman.

Abernathy, William J., and Kenneth Wayne
1974 "Limits of the Learning Curve." Harvard Business Review, September-October: 109-119.

Arden, Bruce W. ed.
1980 What Can Be Automated?. Cambridge, MA, MIT Press.

Banker, Rajiv D., and Chris F. Kemerer
1988 "Scale Economies in New Software Development." M.I.T. Center for Information Systems Research, Working Paper.

Baumol, William J., John C. Panzer, and Robert D. Willig
1982 Contestable Markets and the Theory of Industry Structure. New York, Harcourt Brace Johanovich.

Blau, Peter M., and Richard A. Schoenherr
1971 The Structure of Organizations. New York, Basic Books.

Boehm, Barry W.
1976 "Software Engineering." IEEE Transactions on Computers, C-25, No. 12: 1126-1241.

1981 Software Engineering Economics. Englewood Cliffs, NJ, Prentice-Hall.

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

Chandler, Alfred D. Jr.
1977 The Visible Hand: The Managerial Revolution in American Business. Cambridge, MA, Harvard University Press.

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

Child, John
1972 "Organization Structure, Environment, and Performance: The R o l e of Strategic Choice." Sociology, 1: 1-22.

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

Cusumano, Michael A.
1987a "Hitachi: Pioneering the Factory Model for Large-Scale Software Development." Sloan School of Management, Working Paper #1886-87.
1987b "Toshiba's Fuchu Software Factory: Strategy, Technology, and Organization." Sloan School of Management, Working Paper #1939-87.

1987c "NEC: Standardization Strategy for a Distributed 'Software Factory' Structure." Sloan School of Management, Working Paper #1954-87.

1988a "Software Development Corporation: Defining the Factory Challenge." Sloan School of Management, Working Paper #1887-87, Revised.

1988b "Fujitsu Software: Process Control and Automated Customization." Sloan School of Management, Working Paper #2044-88.

1988c "Hardware and Software Customer Satisfaction in Japan: A Comparison of U.S. and Japanese Vendors," Sloan School of Management, Working Paper, December.

1989 "The Software Factory: A Historical Interpretation." IEEE Software, March: 23-30.

Harvard Business School
1986 "VLSI Technology, Inc. (A): Automating ASIC Design," Case Study 0-686- 128.

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

Hirshhorn, Larry
1984 Beyond Mechanization: Work and Technology in a Postindustrial Age. Cambridge, MA, MIT Press.

Hounschell, David A.
1984 From the American System to Mass Production, 1800-1932. Baltimore, Johns Hopkins University Press.

Jacoby, H.
1973 Bureaucratization of the World. Berkeley, University of California Press.

Johnson, James R.
1989 The Software Factory: Managing Software Development and Maintenance. Wellesley, MA, QED Information Sciences.

Jones, Capers
1986 Programming Productivity. New York, McGraw Hill.

Lawrence, Paul, and Jay W. Lorsch
1967 Organization and Environment: Managing Differentiation and Integration. Boston, MA, Harvard Business School Division of Research.

Lorange, Peter, Michael A. Scott Morton, and Sumantra Ghoshal

1984 **Strategic Control Systems**. St. Paul, Minn., West Publishing.

Mansfield, Edwin
1985 **Microeconomics: Theory/Applications**. New York, W.W. Norton & Company.

March, James G., and Herbert A. Simon
1958 **Organizations**. New York, John Wiley.

Matsumoto, Yoshihiro.
1987 "A Software Factory: An Overall Approach to Software Production." In Peter Freeman, ed., **Tutorial: Software Reusability**, 155-178. Washington, D.C., Institute of Electrical and Electronics Engineers.

Miewald, Robert D.
1970 "The Greatly Exaggerated Death of Bureaucracy." **Californian Management Review**, Winter: 65-69.

Miles, Raymond E., and Charles C. Snow
1978 **Organizational Strategy, Structure, and Process**. New York, McGraw-Hill.

Miles, Raymond E., et al.
1978 "Organizational Strategy, Structure, and Process." **Academy of Management Review**, July: 546-562.

Mintzberg, Henry
1979 **The Structuring of Organizations**. Englewood Cliffs, N.J., Prentice-Hall.

Orlikowski, Wanda
1988 "Information Technology in Post-Industrial Organizations." Unpublished Ph.D. Dissertation, New York University, Graduate School of Business Administration.

Perrow, Charles
1972 **Complex Organizations: A Critical Essay**. Glenview, Ill., Scott, Foresman.

Perrow, Charles
1967 "A Framework for the Comparative Analysis of Organizations." **American Sociological Review**, April: 194-208.

Pfeffer, Jeffrey
1981 **Power in Organizations**. Marshfield, MA, Pitman.

Porter, Michael
1980 **Competitive Strategy: Techniques for Analyzing Industries and Competitors**. New York, The Free Press.

Pugh, Derek S.
1973 "The Management of Organization Structures: Does Context Determine Form?" **Organizational Dynamics**, Spring: 19-34.

Robbins, Stephen P.

1987 Organization Theory: Structure, Design, and Applications. Englewood Cliffs, N.J., Prentice-Hall.

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

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

Stinchombe, Arthur L.
1959 "Bureaucratic and Craft Administration of Production: A Comparative Study." Administrative Science Quarterly, 4: 168-187.

Tabachnick, Barbara L., and Linda S. Fidell
1983 Using Multivariate Statistics. New York, Harper and Row.

U.S. Department of Commerce
1984 A Competitive Assessment of the U.S. Software Industry. Washington, D.C., International Trade Administration.

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

Zuboff, Shoshana
1988 In the Age of the Smart Machine: The Future of Work and Power, New York, Basic Books

NOTES

1. A forthcoming book based on these case studies and other material on the technology and management responses is Michael A. Cusumano, **The Software Factory: Japan's New Challenge in Technology and Management**, New York and Oxford, Oxford University Press, 1990.

2. This list excludes a group of 200 people that did not produce software for sale but made up the System Development department of Hallmark Cards, Inc., which produced programs for in-house use but is referred to by a former manager as a "Software Factory" (Johnson, 1989).

3. Additional questions were also sent to survey participants, although comments from the responders, 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.

4. In the case of Toshiba, a single large facility (approximately 2300 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.

5. 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."

6. Interviews with Anne Smith Duncan, Software Engineering Manager, Software Development Technology, Digital Equipment Corporation, 2/10/88; and Wendy McKay, Project Manager, Educational Software, Digital Equipment Corporation, 12/88.

7. Interviews with Matsumoto Yoshiharu, R&D Department Manager; Matsuzaki Yoshizo, Applications Software Department Manager; and Takahashi Tomoo, Applications Software Department Deputy Manager, Hitachi Software Engineering, 9/3/87.

8. 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. See DATAMATION, 15 June 1987, pp. 28-32. 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).