Shifting Economies: From Craft Production to Flexible Systems and Software Factories

Michael A. Cusumano

Massachusetts Institute of Technology
Sloan School of Management
Working Paper# 3325-91/BPS

Draft: August 26, 1991

ABSTRACT

This article discusses the evolution of production systems from craft or job shops to conventional mass production and then to flexible design and production systems. The major argument is that factory concepts and technologies have been evolving in two directions: First, they have become more versatile in the variety of products they can produce, resulting from innovations in production and design technology as well as management techniques. Second, companies have extended factory-like tools and techniques backward toward design operations, gradually bringing more discipline, automation, and thus efficiency into the realm of engineering work, including the relatively new field of computer programming. The result, in both manufacturing and design, has been a shift from simple scale economies, as in conventional mass production, to scope economies -- efficiencies gained in the design and production of multiple products. An examination of how flexible-factory tools and techniques have been applied in large-scale software development illustrates these trends.

1. Introduction

Factory systems emerged in the 1800s as various industries attempted to move the process of product replication beyond the "craft" stage -- where highly skilled craftsman treated each job and each product as unique -- to a more efficient mode of operations aimed at economies of scale. Rather than making products one at a time, new efficiencies came through standardized and interchangeable parts, fixed and simplified production procedures, use of less skilled workers with divisions of labor, mechanization or automation, and stricter process and quality as well as accounting controls. Factory systems decreased the flexibility[1] of production organizations that could once adapt to any job and make a variety of products, and presented new difficulties in labor relations as employees had to adapt to routinized work. Nonetheless, factories offered an efficient way to make sophisticated goods of standardized quality and low costs, thus bringing an array of products once too expensive into the reach of the average consumer.

This article discusses the evolution of production systems from craft or job shops to conventional mass production and then to flexible design and production systems. The major argument, detailed in Section 2, is that factory concepts and technologies have been evolving in two directions: First, they have become more versatile in the variety of products they can produce, resulting from innovations in production and design technology as well as management techniques. Second, companies have extended factory-like tools and techniques backward toward design operations, gradually bringing more discipline, automation, and thus efficiency into the realm of engineering work. The result, in both manufacturing and design, has been a shift from simple economies of scale, as in the conventional mass production of a limited number of products, to *economies of scope* -- efficiencies gained in the design and production of multiple products. The third section of this article provides an

illustration of these trends by examining how companies have applied factory-like tools and techniques to large-scale software development since the 1960s, beginning with General Electric, AT&T, and System Development Corporation in the United States, and then Hitachi, Toshiba, NEC, and Fujitsu in Japan.

2. Evolution of Design and Production Systems

2.1 Customization Versus Replication

For most products, design and production require several distinct steps or phases, proceeding more or less sequentially, with some iterations and overlaps. Most products begin with an idea generated by manufacturers, suppliers, or consumers that seems to define an existing or future customer need. Companies then experiment with concepts and create a formal concept proposal. Next, this must be translated into a detailed design, and the resulting documentation or blueprints then become prototypes. When the designers and prototype builders are satisfied, the completed design becomes a production model, which can be tested and refined further [42, 58, 72, 78].

The sequence ends with the production model if there is only one job -- one product, one customer. In this case, the product and the process needed to make it might be customized in an organization such as a *job shop* by workers that resemble the craftsmen or artisans who made a variety of products before the advent of mass-production factories. If there is more than one potential customer, a firm can create different types of organizations aimed at replicating this design with varying levels of efficiency and output volumes from a few (*batch production*) to many (*mass production*).

From the customers' point of view, a product is customized only if it is tailored

for their specific needs. From the producers' point of view, however, there are several options. Some new designs can be customized or "made from scratch," while others reuse designs, components, tools, processes, or particular people from other projects. Volume of replication, degree of components or design reuse, and continuity of process know-how, as well as the coordination mechanisms needed to manage design and production tasks, are some variables distinguishing different types of product-development and production approaches. These distinctions are important because academics as well as managers have used them to explain the principles behind both job shops and factories, and thus the logic that suggests what processes and methods of organization seem most appropriate for particular situations.

In the case of software, for example, making a new and especially a large-scale program can be a highly complicated process of design, coding, and testing. Since key development tasks have been difficult to standardize, structure, and automate, many managers have viewed software as most appropriate for a job-shop or craft approach that treats each project as unique and relies on highly skilled engineers. This approach may seem suitable also because replication of a completed program is a simple, almost instantaneous electronic process, requiring no complex manufacturing or assembly operations. On the other hand, when software producers or other kinds of design and engineering organizations encounter a lot of commonality or redundancy from project to project, then it becomes possible to reuse designs or components or institute some level of standardization and automation that aids the organization in development products efficiently in multiple projects. As discussed below, the realization that there are common design elements and engineering methodologies useful for similar but distinct projects has made factory-like tools and techniques possible in software development and other design tasks.

2.2 Interpretations of Production Systems

Modern factories emerged when firms introduced a series of product and process innovations that made possible the efficient replication of a limited number of designs in massive quantities. These facilities captured what have been called *economies of scale* -- efficiencies that come when average production costs per unit of output for a particular product decline as total volumes of production increase, at least up to a certain limit [16, 41, 59]. Specific characteristics of an industry and its environment (degree of divisibility of processing equipment and operations, degree of product and process standardization or complexity) seem to have determined whether firms could or should move to factory-oriented modes of design and production. While factory manufacturing can be highly efficient, usually managers have considered it economically feasible only with high production volumes and limited as well as stable product designs, due to the large capital requirements usually needed to introduce mass-production systems and the time usually needed to change equipment for different product models.

Various authors have discussed these characteristics as well as benefits and limitations of factories and alternative kinds of production processes, including those that preceded mass production. For example, Woodward identified three basic types of organizations or processes: unit or craft production (job-shop and small-batch), large-batch and mass production, and continuous processing operations as in chemical manufacturing [75]. Unit or job-shop production has non-standard inputs and outputs -- i.e. each finished product, and the components that go into it, tend to be different. Therefore, managers are unable to standardize materials or formalize development processes. Job-shop organizations typically require many ad hoc adjustments to make a product; to allow workers to make these adjustments, managers exert control only at a low level, that is, usually by first-line supervisors. It also

seems important for supervisors and workers to be highly skilled -- hence the label *craft* seems appropriate for this mode of operation. In general, in job-shop or small batch production, the lack of standardization in processes and products, and management controls, as well as high skill requirements, make it difficult for managers to find scale economies.

Factory systems replaced craft modes of production as firms learned how to rationalize product designs as well as production work itself. In describing the evolution of factories during the 18th and 19th centuries in Britain, Europe and the United States, initially with products such as textiles, guns, agricultural machinery, and sewing machines, Chandler and Hounshell noted how managers raised worker productivity and lowered unit costs by standardizing and then integrating production processes in large, centralized facilities; developing interchangeable components; closely coordinating the flow of each process; dividing and specializing labor; mechanizing or automating tasks; and imposing rigid managerial, accounting, and other types of controls [15, 27]. Hunt saw this evolution in terms of distinct "ages," such as from craft to machinery, exemplified by the use of steam engines in factory mass production during the 18th century, to the power age after the introduction of electricity [28]. Landes similarly described this transition -- often labelled "the industrial revolution" -- as involving the substitution of mechanical devices for human skills and inanimate power for human or animal strength [39].

But while factory organizations provided higher worker and capital productivity, the nature of the processes, equipment, and worker job routines made it difficult to introduce new products or processes quickly and economically, or to meet the demands of customers with distinctive tastes. Because new technologies or product and processes variations continue to appear in most industries, and because not all customers want commodity products, factory-oriented design and production

systems have never completely replaced craftsman or job shops.

A key topic in the field of production and operations management has thus been this problem of how to match different types of products with different approaches to production or product replication. Hayes and Wheelright [24] as well as Schmenner [62], for example, have refined distinctions found in Woodward and elaborated on the characteristics of different processes as well as the specific product and market characteristics that should accompany each. Abernathy and Utterback have argued as well as that many industries tend to mature in a sort of "life cycle" whereby product innovations tend to decrease over time. This maturation, although by no means certain or uniform in all industries, at times allows firms to take advantage of accumulated experience and introduce process innovations aimed at improving efficiency [1]. At the facility level, product and process standardization has made it possible to move from project or job-shop modes of operation to mass-production or continuous manufacturing (Figure 1).

A common problem, however, is the inflexibility firms often encounter when moving toward large-scale factories and continuous production, because designs and processes became difficult and expensive to change. Abernathy and Wayne [2] used the experience of Ford and its Model T production facilities in the 1920s to demonstrate how a company can face bankruptcy by pushing process rationalization and scale economies too far -- for example, assuming product technology or consumer tastes were more stable than they were, and making investments in expensive and rigid factory systems that took long periods of time to change to accommodate new models or production techniques.

Piore and Sabel [60] have extended this line of argument to assert that mass production in general -- any attempt to use special-purpose machinery and semi-skilled workers to make standardized goods -- is nearly obsolete, as both markets and

workers have come to demand more variety in products and discretion in work environments than mass-production firms can deliver. Their interpretation is that "flexible specialization" -- the use of numerous small, specialized suppliers, relying on relatively skilled workers -- will supplant mass production in large organizations. Other authors have placed less emphasis on suppliers but have also argued that conventional mass production is more or less obsolete in many industries where it once dominated, typified by automobiles, with "lean production" as represented initially by a series of Japanese manufacturing techniques emerging as the new dominant approach. These techniques include more versatile equipment, workers, and suppliers, as well as more manufacturing in small rather than large production lots, which keeps inventories low, catches mistakes quickly, offers customers more product variety, and matches product mixes more accurately with market needs [17, 18, 38, 51, 63, 74].

Following in the tradition of Woodward and later production-management literature, various organizational theorists have concluded not that one approach is obsolete but rather that no one structure is appropriate for all competitive situations and technologies [7, 22, 40, 49, 57]. In other words, as Woodward suggested, specific approaches fit particular conditions better than others. For example, Mintzberg [49] and Borum [12] characterized the environments in which firms operate as a mixture of stable or dynamic, and simple or complex. An environment that is complex and dynamic should be met with organizational responses of "adhocracry" and "mutual adjustment" -- denoting craft modes of operation rather than factory-type structures that would better fit characteristics such as "machine bureaucracy" and work-process standardization (Table 1). If these characteristics determine which products should be made in what organizations, then new and unstandardized technologies, or products with truly unique features and requirements for each job

or user, would seem to be poor candidates for rigid factory modes of production. Software would fall into this category as well, except in applications that, over time, have become relatively routine or contain high degrees of redundancy even in systems that require some degree of tailoring for different users.

Another classification scheme, suggested by Perrow [57], adds further insights into when factory-like discipline or structuring of work might apply to different kinds of production and even product-development operations such as software. In focusing on task variability and the analysis of technical problems, Perrow argued that organizations dealing with "routine technologies" encounter few exceptions and therefore face problems that, over time, become possible to analyze and solve through formal procedures or tools. A standardized process then eliminates the need to have large numbers of highly-skilled (and usually expensive) professionals capable of re-inventing solutions each time problems or projects occur. Firms dealing with what Perrow called "engineering technologies" encounter more exceptions, but the problems are still relatively well-defined and manageable systematically. Perrow contrasted routine and engineering technologies with "craft technologies," defined by problems of a limited range but ill-defined and therefore difficult to analyze, as well as "non-routine technologies," which required many exceptional and difficult tasks (Table 2).

The literature thus offers a variety of historical and organizational explanations for different types of production organizations as well as specific product-process typologies for both production management and more intellectual forms of work, including design and engineering tasks or even service operations. A common theme in various literatures is that firms need to find an appropriate match, which in essence means a proper balance between efficiency and flexibility. Factory systems and counterparts in engineering or service businesses may offer a more efficient means of

managing operations than in craft-like organizations, but with limitations on the versatility of the organization in the sense of being able to accommodate variety or exceptions.

Table 3 presents a stylized comparison of the basic strategies, characteristics, and tradeoffs that might accompany a conventional factory and craft or job-shop type of organization. It is clear that mass-production factories, or their analogies for engineering operations, are not appropriate for all types of products or competitive strategies; they have traditionally worked best for limited numbers of designs suited to mass replication and mass consumption. The craft approach, typified by the job shop, offers a less efficient process at least for commodity products but remains necessary for technologies that are still new or emerging, and continues to serve specific market niches, such as for tailoring products for individual needs.

2.3 Flexible Design and Production

Interpretations of factory production cited above assume that organizations adopting mass production accept an inherent tradeoff -- more efficiency for less flexibility, particularly in product variety (see Figure 1). But, in several industries, new managerial and engineering techniques, along with advanced computerized technologies, have been reducing this tradeoff of flexibility for efficiency. The general effect has been to shift competitive emphases from either low-cost production or product differentiation to making different products at low cost. This is achieved not through simple economies of scale based on the mass replication of a small number of designs, but through scope economies based on efficiencies in the design and production of a variety of products in individual production or engineering facilities [9]. This shift has required both (1) the introduction of more flexibility into production processes concerned with product replication, and (2) the introduction of

more efficiency into design and engineering processes concerned with product customization.

Several examples illustrate the kinds of tools and techniques that have facilitated this evolution. Many firms have found ways to design products from component modules that can be configured in different ways, group together similar components to gain efficiencies in engineering and production, decrease the set-up times required for production equipment to change over to different products, and employ workers and suppliers that are relatively skilled and able to make a variety of end products in relatively small volumes rather than in massive quantities but with high levels of productivity and quality. Japanese automobile producers led in the introduction of this approach to manufacturing, which differs from conventional mass production, although producers of machine tools, semiconductors, and other products have used similar approaches to combine job-shop flexibility in end-product variety with the productivity, standardized quality, and low cost structure of automated factories [3, 30].

An important concept for both manufacturing and design used in various industries, called "group technology," involves putting together similar parts, problems, or tasks to facilitate scheduling of parts production, arranging factory layouts, or rationalizing product design and engineering. The underlying principle here is clearly economies of scope rather than of scale, at least as measured by the number of identical components produced through a given process. Critical to any group-technology scheme is a coding system, similar to library references, that makes it possible to classify characteristics related to manufacturing, engineering, purchasing, or other functions. In engineering, parts might be classified by geometric similarities, and matched with process plans and machines capable of making parts to those specifications. Or designs for certain categories of parts might be

coded and filed, so that engineers would have ready access to old drawings and not have to "reinvent the wheel" more than once. Advances in computer processing capacity and in programming sophistication have facilitated development of the coding and retrieval systems needed for group technology to work well for large numbers of items, leading to extensive monetary savings in firms. The basic concept, however, is as old as interchangeable parts [29].

Computer-aided tools for design and engineering have been particularly useful for efficiently designing and replicating a variety of products from standardized parts, with much of the design knowledge incorporated into computer programs or databases, rather than relying exclusively on very highly skilled people. Many computer-aided tools employ coding schemes that rely on what are, essentially, group-technology concepts, and provide semi-automated support for the selection and modification of standardized components. Integrating computer-aided design tools with flexible computer-aided or computer-integrated manufacturing systems (variously referred to as CAD/CAM, FMS, CIM) makes it possible to test different design and processing ideas on a computer screen and then transfer completed digitalized designs to automated manufacturing tools, with little or no penalty associated with low production volumes, because automation largely eliminates costly labor-intensive processes. The result is that customers do not have to pay high prices for fully customized products; nor do they have to put up with standardized commodity products that are inexpensive but do not fulfill their needs satisfactorily.

An important demonstration of this capability is in application-specific integrated circuit (ASIC) design, one of the fastest growing areas of the semiconductor industry. One method is to mass produce gate-array chips, which contain transistors laid out in fixed rows; a designer then uses an automated tool which follows programmed design rules to create "custom-routed" connections to fit

different applications. Another approach is to develop standard cells, which contain small logic combinations, which can be configured in different ways. At even higher level of abstraction and standardization is the alternative of building standardized "megacells" such as graphics controllers, arithmetic logic units, or microprocessors, that, using computer-aided design tools, can be configured easily into different chip designs. CAD tools can select routing combinations or configurations on the basis of programmable rules, as well as perform simulations and testing functions to aide the design process. VLSI Technology, Inc. (VTI) in the United States has even standardized around design tools simple enough for non-specialists to use, and created what management refers to as an ASIC "design factory":

> Design engineers are the bulk of the work force now at VTI's dispersed design centers.... One needs rudimentary knowledge of how to use the work station software -- as a CAD or drafting tool -- to copy schematics onto the system. It requires care and attention to detail -- but not extensive engineering training. A technician with a high school or associate degree is adequate. We can save engineering time for better use, and partition the design "production" process into a more elaborate division of labor, utilizing both computer and human resources more efficiently. *Work can move along like in a factory*, from person to person, task to task. The manager maintains line balances by allocating human and computer resources [italics added] [31, p. 15].

In short, evidence from various fields indicates that factories no longer need to be rigid manufacturing organizations, while engineering and design work no longer needs to be done completely by hand. Craft or job shops may still be most appropriate

for totally new or completely unstandardized products, very low-volume products, or unique market segments. But new concepts of manufacturing and design, supported by computer-aided technologies, have made it possible to combine flexibility and efficiency in factories as well as engineering organizations. Table 4 summarizes some of the features one might expect in a design and manufacturing system emphasizing at least temporarily standardized but evolving processes and tools, modularized components, and the production of semi-customized products. If integrated in a single organization, this approach might describe a *flexible design and production system* capable of offering customers a variety of products at potentially lower prices than traditional craft production.

A simple way to conceptualize the distinctions among craft or job-shop organizations, conventional factories, and flexible design and production systems, is to visualize a spectrum reflecting how much emphasis each approach places on tailoring products or production processes to specific customers or market segments. In practice, the spectrum is probably continuous, but it can be thought of as containing three basic options: (1) **Full Customization:** design and manufacture unique products, where each is different and each process -- tools, components, specific work rules, team members -- are also different. (2) **Semi-Customization:** design a semi-customized (or semi-standardized) product, where a standard procedure is to reuse some or all components from stock and configure them in different ways for different customers. (3) **Full Standardization:** design and manufacture a commodity-type product, where components are interchangeable, or where components and final products are mass produced, using a standardized process.

Full customization and full standardization apply most directly to industries clearly segmented among users who desire either fully tailored products or low-priced

commodities. But even if customers demand different products, companies might still find they can recycle at least some designs or components for some period of time, as well as reuse procedures, controls, tools, tests, and documentation. Therefore, unless each product, and the process and people required to design and build it, are entirely different, semi-customization, benefitting from at least some economies of scope, if not scale, should be possible. Furthermore, even firms making commodity products for the mass market can choose to reuse components or other inputs. Thus semi-customization is a product-process option that firms seeking to meet the needs of a broad range of customers might select.

A firm might introduce semi-customization and a flexible design and production system where there is a need for professional-level skilled workers but with at least some stability or predictability in customer requirements or product designs. Semi-customization as a product-process option would thus allow the firm to capture repeatable elements for some period of time, while allowing for the addition of new components, tools, or processes to meet the distinctive needs or customers or to accommodate incremental changes in technology or markets. In these cases, producers face what Perrow viewed as engineering technologies: some exceptions in product requirements but relatively well-defined tasks, perhaps because of similar customer needs but also reflecting management efforts to create permanent groups of people experienced in particular applications and to channel similar work to these focused facilities. The challenge here would be to standardize temporarily -- leaving room for at least incremental evolution -- basic tasks and, especially, the knowledge or tools needed to deal with a range of common problems; and divide labor only for clearly defined tasks when it is necessary to leverage the most skilled employees across multiple projects. The result is to de-skill work or formalize rules and procedures only in a relative sense, compared to modes of operations before managers

explicitly adopted factory strategies or to implicitly ad-hoc approaches with little or no formal structures. These management policies would combine some of the flexibility of craft or job shops, in that they remained adaptable enough to make unique or customized designs as well as to evolve, with some of the efficiencies of engineering and factory production, but based on scope rather than scale economies across a series of projects.

The typology and organizational options should be available in any market where there is some segmentation and at least some temporary standardization of needs, basic technologies, and development processes. Taking the example of software development, rather than viewing this as a single market requiring a single process, such as the craft approach, one can define a high-end custom segment, such as for unique defense or plant-automation systems. Producers focus on optimizing product performance and process flexibility; they experience few economies of scale or scope, at least for totally new jobs, but customers usually pay high premiums. On the other end of the spectrum, such as for standardized programs like word-processing packages, firms need to produce "best-selling" products suitable for many users; therefore, at least for the first generation of a product, may not want to try to structure the development process. With a successful effort, however, they can maximize economies of scale -- not in development but in program replication, achieving profits through small margins on high sales volumes. There is also a middle-segment to the market, where the "factory" approach and "semi-custom" best characterize both products and processes. Some customers are sensitive to combinations of price and performance; producers, as a result, attempt to exploit commonality or automation across multiple projects, offering more customization than fully standardized systems and less unique features, but lower prices, better reliability, or faster delivery than fully customized systems. (Table 5).

3. The Case of Software[2]

Software factories emerged within the computer industry as producers attempted to push forward the state of programming practice in order to move beyond loosely organized craft or job-shop modes of operation that treated each project as unique. The result was not conventional mass production and scale economies, since product replication is a simple electronic process, the key tasks in software work consist of activities such as design and testing, and even software products reusing large amounts of components from other systems still contain unique or customized features. Software factories came to resemble flexible design and production systems aimed at scope economies achieved through managing multiple design or engineering projects systematically, rather than treating each project or job as unique. This approach has relied on permanent development groups dedicated to particular families of products (rather than forming and disbanding groups at the completion of each project), process R&D groups that develop or refine standardized methods and tools (software programs and databases that facilitate the writing of other software), planned rather than "accidental" or ad hoc reuse of program components and designs, some divisions of labor to leverage the most highly skilled people across multiple projects, common training programs, and disciplined procedures for project management as well as product quality control.

Some firms in the United States and Europe have introduced these and other concepts for large-scale programming operations, but in varying degrees. IBM was the leader in the United States after it had to organize the development of several operating systems for its innovative System 360 family of computers during the 1960s and then for the next generation System 370 machines. IBM did not explicitly adopt the factory analogy, however, and continued to allow programming centers and

individual projects to operate with large degrees of autonomy, especially for applications programs [19]. The discussion here focuses on the specific origins of the term "factory" as used in software and the histories of several companies and facilities that have explicitly used this concept to manage software development.

3.1 The Development Process

Software programming, even in factory environments, is not like conventional manufacturing but more closely resembles design and engineering. It is essentially a process of analysis and translation: analyzing a problem and then breaking it down into a series of smaller tasks expressed in a manner ultimately understandable to a computer. Software developers 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 assemblers and compilers usually perform this transformation automatically [5, 61].

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 (functional) design. For a product that continues in service with periodic modifications, post-delivery maintenance may become by far the most costly activity, consuming up to 70% or so of total expenditures over its lifetime, according to commonly-cited data on life-cycle costs [11]. New generations of computer-aided software engineering (CASE) tools are

also automating increasing aspects of the development process and changing these life-cycle costs, although most software development in the early 1990s, especially new and large projects, remained a complex, labor-intensive, and expensive process.

One reason is that, while most software projects go through similar phases that appear sequential, as in other kinds of product development, the design and production process is often highly iterative, requiring developers to go back and forth among requirements analysis, specification, program design, coding, testing, redesign, re-testing, and so on. Experienced engineers 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 software development something less than an exact discipline. This is especially true since project requirements might contain new functions that are difficult to build, customers often change their minds about features they desire, and individuals usually take different amounts of time to perform similar operations. Furthermore, the larger projects become, the more interdependent activities and components they require, and the greater the uncertainty of the final product's cost, schedule, and performance.

Huge variations in personnel performance -- as much as 26 to 1 in studies comparing the programming productivity of the best people to the worst [32, 55] -- appear to stem from differences not simply in native ability, but also in the particular experiences of the individual. Someone 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 a program must perform in each application greatly affect the amount of thought, time, and experience required to solve design problems and write or test 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 -- strategies common in factory production or in highly structured engineering operations. As a result, loosely-structured job-shop approaches, with people, tools, methods, procedures, and designs changed, if necessary, for each project, remain highly suitable for many software projects.

This does not mean, however, that the craft approach is best for *all* kinds of software projects. To the contrary, job-shop 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. Various software producers have in fact identified what problems they commonly face and then built an infrastructure -- of tools, methods, reusable components, and permanent groups of people, not unlike the approaches of factory organizations or disciplined engineering departments in other industries -- to manage these problems more effectively.

Factory-like approaches that resemble flexible design and production systems (rather than craft approaches or conventional mass production) became the dominant methodology for software development at Japan's largest computer manufacturers during the 1970s and 1980s. They became popular because of several conditions in the Japanese market. First was the enormous demand for labor-intensive custom applications that had high levels of redundancy among projects, estimated at up to 90% across delivered programs [6, 35]. The Japanese also suffered from a severe shortage of skilled programmers since the 1960s and continuing through the present

[4, 25, 56]. Furthermore, Japanese producers tend to follow U.S. product standards and concentrate not on software invention but on improving the programming process and the "price-performance" of computer systems they deliver. The Japanese have also recognized that, as in other industries, customers are sensitive to a combination of price, delivery, and reliability, as well as features. These conditions made the benefits of a flexible factory approach -- more predicable project and quality control, high system reliability, cheaper prices, and quicker delivery, based on the recycling of proven designs -- appealing to both Japanese producers and consumers [20, 23, 34, 66, 67, 70, 71].

3.2 Early U.S. Proposals and the SDC Factory

The earliest public proposals for the introduction of explicit factory methods, tools, and management systems to software development appeared in the late 1960s, as outgrowths of comparisons of programming with established engineering and manufacturing processes. An engineer at General Electric, R.W. Bemer, made numerous suggestions that culminated in a paper encouraging GE to develop a software factory to reduce variability in programmer productivity through standardized tools, a computer-based interface, and an historical database for financial and management control. GE's exit from the computer business in 1970 ended the company's commitment to commercial hardware and software production, although Bemer provided the industry's first working definition of what might constitute a software factory:

> [A] software factory should be a programming environment residing upon and controlled by a computer. Program construction, check-out and usage should be done entirely within this environment, and by using

the tools contained in the environment... A factory...has measures and controls for productivity and quality. Financial records are kept for costing and scheduling. Thus management is able to estimate from previous data... Among the tools to be available in the environment should be: compilers for machine-independent languages; simulators, instrumentation devices, and test cases as accumulated; documentation tools -- automatic flow-charters, text editors, indexers; accounting function devices; linkage and interface verifiers; code filters (and many others) [10, pp. 1626-1627].

While Bemer focused on standardized tools and controls, Dr. M.D. McIlroy of AT&T emphasized another factory-like concept -- systematic reusability of code when constructing new programs. In an address at a 1968 NATO Science Conference on software engineering, McIlroy argued that the division of software programs into modules offered opportunities for "mass production" methods. He then used the term "factory" in the context of facilities dedicated to producing parameterized families of software parts or routines that would serve as building blocks for tailored programs reusable across different computers [48]. But reception to McIlroy's ideas was mixed: 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. Software was also heavily dependent on the specific characteristics of hardware. Nor did anyone know how to catalog program modules so they could be easily found and reused [26]. Nonetheless, by the late 1960s, the term factory had arrived in software and was being associated with computer-aided tools and management-control systems, as well as modularization and reusability.

In fact, one of the U.S. leaders in the custom software field, System

Development Corporation, formerly a part of the Rand Corporation and in 1991 a Unisys division, established the first U.S. software facility called a factory in 1975-1976. SDC had been separated from Rand in the 1950s to develop the SAGE missile control system for the U.S. Department of Defense. It later took on other real-time programming tasks as a special government-sponsored corporation, but went public in 1970. Top management then had to control software costs and launched a process-oriented R&D effort in 1972 to tackle five problems SDC programmers continued to encounter project after project: (1) Lack of discipline and repeatability or standardized approaches to the development process. (2) Lack of an effective way to visualize and control the production process, as well as to measure before a 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 [13, 14].

After several years of R&D work, a team of SDC engineers, led by John B. "Jack" Munson, constructed a detailed factory plan that consisted of three elements: an integrated set of tools (program library, project databases, on-line interfaces between tools and databases, and automated support systems for verification, documentation, etc.); standardized procedures and management policies for program design and implementation; and a matrix organization, separating high-level system design (at customer sites) from program development (at the Software Factory). The

first site to utilize the factory system, which SDC copyrighted under the name "The Software Factory," was a facility of about 200 programmers in Santa Monica, California. SDC thus continued to have "program offices" at each customer site, with managers that maintained responsibility throughout the life-cycle for project management, customer relations, requirements and performance specifications, systems engineering, and quality control and assurance. To build the actual software and test it, however, program managers that wanted to use the factory (its usage was not mandatory) had to transfer system specifications to what was essentially an assembly line of three groups within the new software factory, which served SDC's System Division: Computer Program Design, Computer Program Development, and System Test and Verification.

SDC gave the name Factory Support System to the "basic structural and control components" designed to facilitate the factory methodology. This tool set, written in a high-level language to ease portability, ran on an IBM 370 mainframe computer and used the facilities of IBM's operating system to automate procedures for keeping track of program development and collecting data. Tools included compilers and other basic programs that worked with the operating system, as well as Top-Down System Developer (TOPS), a modeling tool that helped outline and verify designs as well as describe much of the control and data interface logic in the actual coding language; Program Analysis and Test Host (PATH), which analyzed a source program and inserted calls to a recording program at appropriate locations, helping developers find information about the structure of the program to aid in testing; Integrated Management, Project Analysis, and Control Techniques (IMPACT), which utilized production information on milestones, tasks, resources, system components, and their relationships to provide schedule, resource computation, and status reports at the individual components level or summarized at any module or task hierarchy level.

It took a year and a half during 1975-1976 for the R&D team to identify standards and procedures -- general rules and specific guidelines -- that might be applied to a variety of software projects. They based their process around a life-cycle model of software development covering the major activities, events, and product components common to all projects. The methodology, codified in what SDC called the Software Development Manual or SDM, called for structured design and coding, top-down program development, and program production libraries. In addition, SDM outlined a management and control process, providing guidelines for planning, project control, review and evaluation procedures, and quality assurance. The R&D team in part borrowed from existing U.S. military standards, but established most of the SDM methodology by examining previous projects SDC had done through written records and interviewing personnel to determine what had worked well and appeared to represent "best practice" within the company. According to two of the key factory architects, Harvey Bratman and Terry Court, this effort was critical to creating a common language and methodology that made the factory more than just a building with programmers working from a common pile of tools.

Approximately 10 projects went through the SDC Software Factory between 1976 and 1978, including systems for weather-satellite control, air defense, and communications for the Los Angeles Police Department. SDC managers claim that all the projects, with the exception of the system for the L.A. Police, came in on time and within budget, and with fewer defects and problems than SDC usually experienced with large systems. The company history even described the factory projects as, for the most part, "accurate, timely, and on budget," and models of "optimum software development" [8, pp. 205, 224]. In fact, the factory worked so well that SDC's chief executive, George Mueller, directed all divisions to adopt the methodology as a corporate standard, and in 1978 he promoted Munson to oversee this transfer.

After Munson left, however, the factory fell gradually into disuse. Tools were not very portable among different projects; program managers preferred to build their own software, rather than hand over specifications to the factory; and specifying the L.A. Police system, which was an unfamiliar application for SDC engineers, took a year or more longer than scheduled, leaving factory programmers idle as the number of other new projects coming into the facility declined. Future jobs also went back to the project system, rather than using the factory structure for program development. The factory methodology remained through the SDM manual, although SDC dispersed the factory personnel among different projects and sites. SDC also updated the manual every few years and continued to use it through the late 1980s, while concepts from the factory methodology spread to other firms after the U.S. Department of Defense contracted with SDC in 1976 to devise guidelines for military-use software procurement. SDC completed a first set in 1979, with the help of the U.S. Department of Defense and an offshoot of MIT's Lincoln Laboratories, Mitre Corporation; the government subsequently published these procedures as a 16-volume set of guidebooks on software acquisition and management [8]. Perhaps most important, SDC also influenced factory initiatives underway or soon to appear in Japan.

3.3 Hitachi: Organizing for Process and Quality Control

Hitachi boasted the oldest and largest software factories in Japan (Table 6). Its Software Development Center (formerly the Software Works), established in 1969, had approximately 4000 personnel in 1991, including approximately 2000 assigned from Hitachi subsidiaries and subcontractors. This built a range of basic software for Hitachi mainframes, including operating systems, language compilers, and database systems. The Information Systems Development Center (formerly called the Omori

Works and then the System Design Works), separated from the original factory in 1985, housed 7000 software developers (including at least 4000 personnel from subsidiaries and subcontractors) in two 31-story towers. This facility, unlike some other Japanese applications software factories, combined systems engineers (those who designed the systems) with programmers that built the actual software (did the module design, coding, and testing). It concentrated on customized business applications such as for financial institutions, securities firms, inventory control, management information, accounting, and personnel management.

By founding its Software Works in 1969, Hitachi was the first company in the world to apply the term factory (actually, its Japanese equivalent, *kojo*, translated either as "factory" or "works") to an actual software facility [25].[3] A history of independent factories for each major product area prompted executives in the computer division to create a separate facility for software when this became a major activity. The factory represented a deliberate attempt to transform software from an unstructured "service" to a "product" with a guaranteed level of cost and quality, using a centralized organization to achieve productivity and reliability improvements through process standardization and control. Management saw a need to offset a severe shortage of skilled programmers in Japan and deal with numerous complaints from customers regarding defects in the software Hitachi was providing (most of which, along with the hardware, Hitachi was importing from RCA until 1970). It was also important that all Hitachi factories had to adopt corporate accounting and administrative standards; these forced software managers to analyze the software development process in great detail and experiment with a series of work standards and controls that led to the current factory organization.

Managers concentrated initially on determining standards for productivity and costs in all phases of development, based on data collected for project management and

quality control. Hitachi then standardized design around structured programming techniques in the early 1970s, and reinforced these with training programs for new employees and managers. This approach reflected an attempt to improve average skills through a standard process, rather than specify every procedure to be performed in each project and phase of development.

Yet Hitachi managers underestimated how difficult implementing factory concepts such as reusability and process standardization would be. Two examples illustrate this. First, their attempt in the early 1970s to introduce a "components control system" for reusability failed, because of the lack of knowledge about how to produce reusable modules. Managers changed their priorities and decided to find a way to standardize product designs, and then worry about components. A survey of the programming field suggested that structured design and programming techniques would help standardize software design and coding. A committee then spent several years studying these techniques (as they were evolving) and analyzing programs Hitachi had already written. This was truly a pioneering move, because it would be several years before articles in industry journals began discussing structured design and programming widely and companies such as IBM adopted these practices for their internal standards.

Hitachi also failed to introduce one standardized process for both basic software and custom applications software. At the start of the factory, a committee for work standards took on the task of establishing procedures for all activities, based on a life-cycle model of development. They met almost weekly, studying available materials on software development and examining practices within Hitachi, and completed a first-cut set of standards by the end of 1970, referred to as the Hitachi Software Standards (HSS). These covered product planning, design and coding methodology, documentation and manuals, testing, and any other activities necessary to complete

a software product. Although Hitachi managers clearly recognized these procedures and standards would have to evolve as personnel and technology changed, and they made provisions to revise performance standards annually, drawing up the procedures helped them identify best practices within the company and within the industry for dissemination to all projects and departments. Struggling with work standards also helped the committee recognize the need to distinguish between basic systems and applications software, rather than continue trying to impose similar controls and expectations on all types of software development. Hitachi started developing separate standards for applications during 1971-1972 and completed an initial set by 1975 termed HIPACE (Hitachi Phased Approach for High Productive Computer System Engineering). This standardized formats for proposals, designs, and program construction, as well as aided in developing tools for design automation.

By the late 1970s, Hitachi had succeeded in organizing its software factory around a combination of manual engineering and factory techniques -- structured design and coding coordinated with data collection and standard times for each activity, detailed documentation, design reviews independent of project personnel, rigorous defect analysis, and other elements. Only at this point, after spending years studying and standardizing the development process, was Hitachi management ready to invest heavily in computer-aided tools, relying on engineers mainly from the Software Works and the Systems Development Laboratory, part of the central laboratories.

The tools Hitachi devised supported major functions and activities. For basic software, during the late 1970s, Hitachi introduced CASD (Computer-Aided Software Development System) to facilitate design, coding, documentation, and testing, and then CAPS (Computer-Aided Production Control System for Software) for manpower estimation, process flow control, and quality control [33, 64]. Both have also been

continually evolving. For custom applications, in the late 1970s, Hitachi began developing another set of tools under the label ICAS (Integrated Computer-Aided Software Engineering System) [36]. The most important consisted of the SEWB (Software-Engineering Workbench), which supported both system design and programming on advanced work stations, and EAGLE (Effective Approach to Achieving High Level Software Productivity), which ran on Hitachi mainframes and helped programmers build software from reusable modules as well as structure new designs and code for reuse. HIPACE continued to serve as the basic methodology used with these tools.

Performance improvements were impressive. While direct productivity data is unavailable, sales per employee at the Software Works, combining systems and applications programs, doubled after the first year of founding the factory in 1969 and overall rose 12-fold between 1969 and 1984. The percentage of projects delivered late to the Quality Assurance Department dropped from over 72% in 1970 to a low of 6.9% in 1974 and averaged about 12% between 1975 and 1985. Defects reported by users per month for each computer in the field also dropped from an index of 100 in 1978 to 13 in 1983-1984 [19, pp. 184-191]. These process improvements, which were similar at Toshiba, NEC, and Fujitsu, came despite dramatic increases in the kinds of programs Hitachi (as well as its competitors) produced, starting with rudimentary operating systems to a full slate of basic software and applications ranging from programs for banks, factory automation, and management information systems, to packages of tools for customer programming and maintenance support.

3.4 Toshiba: Linking Productivity and Reusability

Toshiba created a highly disciplined factory around focused product lines, using a centralized software facility to develop real-time process control software for

industrial applications [44, 45, 46]. The decision to establish a software factory stemmed from rapid increases in actual and projected demand, beginning around 1975, for industrial control systems relying on a new generation of relatively inexpensive minicomputers. Typical of the changing demands Toshiba faced as sales of its control minicomputers rose were orders from Japanese power-utility companies to develop automated thermal-power generating stations. These used enormous and growing amounts of software; the typical power-generation control system rose from a few hundred thousand lines of code in the mid-1970s to two million by the early 1980s, necessitating years of development and hundreds of new programmers. Furthermore, to achieve safe and untended operation, the hardware and software for these and many other control systems had to be nearly free of defects or at least highly tolerant of system faults.

An R&D group responsible for industrial systems software in Toshiba, led by Dr. Yoshihiro Matsumoto, introduced an organization and process in 1977 integrating tools, methods, management and personnel systems with a physical layout for work stations (Table 7). The strategy for utilizing this infrastructure centered around four policies: (1) standardize the development process, to reduce variations among individuals and individual projects; (2) reuse existing designs and code when building new systems, to reduce redundant work and maximize productivity; (3) introduce standardized and integrated tools, to raise the performance levels of the average programmer; and (4) provide extensive training and career-development tracks for programmers, to relieve the shortage of skilled engineers. Matsumoto initially applied this strategy to power-systems software, but gradually extended the factory approach to all the kinds of systems built within the Fuchu Works.

Perhaps the most delicate feature of Toshiba's Factory was its organizational structure, a matrix imposed over product departments from several operating groups

and divisions, all located on one site, Toshiba's Fuchu Works, located in the western outskirts of Tokyo. The Fuchu Works in 1991 had at least 8000 employees working primarily in four areas: Information Processing and Control Systems, Energy Systems, Industrial Equipment, and Semiconductors (Printed Circuit Board Division). Operating departments within the divisions corresponded roughly to 19 product lines, including systems for information and control in public utilities, factories, power-generation plants, and various industrial and transportation equipment. Each department contained sections for hardware and software design as well as for manufacturing, testing, quality assurance, and product control.

Similarities in the type of software the Fuchu Works built from project to project allowed Toshiba to deliver "semi-customized" programs that combined reusable designs and code with newly written modules, rather than writing all software from scratch. Toshiba also relied heavily on a standardized tool and methodology set, the Software Engineering Workbench (SWB), developed gradually after 1976 and modelled after AT&T's UNIX Programmers Workbench. Toshiba utilized its customized version of the UNIX operating system as well as a full complement of tools for design-support, reusable module identification, code generation, documentation and maintenance, testing, and project-management. Important features of the Toshiba methodology were the design of new program modules (ideally limited to 50 lines) for reusability, the requirement that programmers deposit a certain number of reusable modules in a library each month, and the factoring in of reuse objectives into project schedules and budgets [47].

Software productivity at the Toshiba Software Factory rose from 1390 delivered equivalent-assembler source lines or EASL per person per month in 1976 to over 3100 by 1985, the latest year for which Toshiba has made data publicly available. Meanwhile, reuse levels (lines of delivered code taken from existing software)

increased from 13% in 1979 to 48%. The 3130 lines of EASL source code per month per employee translate into approximately 1000 lines of Fortran, the most common language Toshiba used in 1985 -- considerably more than the 300 lines or so of new code per month commonly cited for U.S. programmers making similar real-time applications. Quality levels (defined as the number of major faults detected after final testing) also improved dramatically after the opening of the factory, ranging from 7 to 20 per 1000 lines of delivered code converted to EASL to 0.2 to 0.05 in 1985 (the range depended on quality-control practices as well as the reliability requirements and the amount of testing customers contracted for) [34, 19].

Toshiba data indicated that reusability was the major reason for productivity and quality improvements. The organization Toshiba created to promote reuse and overcome short-term concerns of project managers and development personnel (such as the longer time required to write and document reusable software) relied on Software Reusing Parts Steering Committees and a Software Reusing Parts Manufacturing Department and Software Reusing Parts Center. The factory formed a steering committee for different areas (with different members, depending on the application) to determine if customers had a common set of needs suitable for a package, and then allocated funds from the Fuchu Works' budget for these special projects. Some packages were usable in different departments, although most served specific applications. The Reusing Parts Manufacturing Department and Parts Center evaluated new software (and documentation) to make certain it met factory standards; after certification, engineers registered the software in department or factory reuse databases (libraries). Registered items required a key-word phrase to represent the functionality of the part or correspond to a specific object, as well as reuse documentation that explained the part's basic characteristics.

Management also relied on an integrated set of incentives and controls to

encourage project managers and personnel to take the time to write reusable software parts and reuse them frequently. At the start of each project, managers agreed to productivity targets that they could not meet without reusing a certain percentage of specifications, designs, or code. Design review meetings held at the end of each phase in the development cycle then checked how well projects met reuse targets, in addition to schedules and customer requirements. At the programmer level, when building new software, management required project members to register a certain number of components in the reuse databases, for other projects. Personnel received awards for registering particularly valuable or frequently reused modules, and they received formal evaluations from superiors on whether they met their reuse targets. The SWB system, meanwhile, monitored reuse as well as deviations from targets at the project and individual levels, and sent regular reports to managers.

3.5 NEC: A Multi-Product, Multi-Process Network

The first step toward a factory structure for software development at NEC consisted of founding the Basic Software Development Division at its Fuchu Works in 1974, thereby separating organizationally operating-systems development from hardware development. NEC subsequently established other organizations at Mita, Tamagawa, and Abiko, all within the Tokyo metropolis or suburbs, for its other software needs. It also dispersed programming work throughout Japan through numerous subsidiaries and satellite offices [21]. However, the separation and separate histories of these facilities gradually presented greater problems for managers pursuing standardization and common goals such as quality improvement throughout the NEC group. Table 8 and the discussion below summarizes the key initiatives NEC managers adopted to create a more effective multi-product, multi-process factory network.

The Software Strategy Project, started in 1976, attempted to integrate programming operations on a group-wide basis (including all in-house divisions and subsidiaries, rather than just the computer division). The objective was to introduce standards for tools, procedures, and methodologies for all phases of development and all aspects of management. Yet it took several years of trial and error to accomplish this while allowing individuals, projects, and divisions sufficient flexibility to tailor standards to the needs of their products and customers. When the Software Strategy Project ended, managers who worked on the effort, led by Dr. Yukio Mizuno, noticed another weakness in NEC's structure for managing software development: the lack of permanent staff to explore and follow through on key issues or technologies. Thus, to ensure continuity and proceed beyond the Software Strategy Project, NEC in 1980 established the Software Product Engineering Laboratory to lead the company's efforts in software engineering R&D, making this organization part of NEC's central research laboratories. The Software Factory Design Project, started in 1986 under the auspices of the laboratory, developed guidelines for designing actual software factories, from higher-level concepts, such as tool and methodology standardization, to smaller details, such as how to arrange programmers' work spaces or recreation areas.

NEC's Software Quality Control (SWQC) Program dates back to 1978, when a handful of NEC managers established a software quality-control study group. Several specific conclusions came out of their reviews. First, research in software development indicated a strong correlation between quality and productivity, reflecting the manpower needed to fix defects. Hence, they concluded that any revolution in software productivity would require correspondingly dramatic improvements in quality-control practices. Surveys of NEC projects also supported the observation that "human factors," i.e. differences in programmer skills and

experience, seemed to be the most important elements influencing individual performance, and that NEC had to address training more seriously if it were to make major advances in productivity or quality [50]. NEC management then set up a quality-control program that focused on motivation, teamwork methodologies, training, and other factors affecting individual performance. Since evidence from manufacturing departments indicated that bringing employees together in small groups helped solve quality problems, NEC imported the concept of quality circles. Next, in 1981, NEC created a formal, company-wide organization covering all aspects of software production, management, services, sales, and training, under the SWQC (Software Quality Control) Program.

The Software Problem Strategy Project, another three-year effort launched in 1982, attempted to encourage more standardization in development and quality-control practices, explore various productivity-improvement measures, and establish or formally designate a series of software factories to serve NEC's different product divisions. Under this project, NEC executives decided to act in three areas. First, they carried out a "software production mapping" that consisted of constructing a logistical and organizational layout of programming operations within NEC by product (basic software, industrial systems, business applications, transmission systems, switching software, and microcomputer software), to determine which software houses NEC's product divisions were using to assist in development and whether divisions needed more help, such as new subsidiaries that might serve as additional software factories. Second, they formalized and systematized procedures for managing software subcontractors. Third, they launched another effort to improve and link software productivity and quality-assurance measures by establishing a Software Productivity Committee to study documentation control, quality control, software productivity and quality measurements, cost estimation, personnel education, project

management, support tools, and production environments.

Although NEC has not released as much performance data as Hitachi or Toshiba, NEC did report major improvements in productivity through reusability in business applications programming as well as significant gains in quality [43]. The SWQC Program, for example, claimed to have achieved declines in defects reported for transmission control software from an average of 1.37 faults per 1000 lines of code to 0.41. In minicomputer operating-system software, the decline in defects was from 0.35 per 1000 lines to 0.20 [50].

On the other hand, the centralized laboratory for software-engineering process R&D did not work quite as well as NEC managers had hoped. Some laboratory researchers had become too "academic" in orientation while engineers and SWQC teams in the product divisions seemed to be doing more useful applied studies. To encourage more practical research that better met the needs of divisions, but without eliminating all basic research, a 1987 reorganization moved the basic researchers to NEC's Central Research Laboratories. This involved no organizational change, since the Software Product Engineering Laboratory had been a part of the central labs. However, physically removing the more academic researchers left a group more concerned with applications of new methods and tools. Management then expanded the number of applied researchers and divided them into four areas under the umbrella of a newly created C&C (Computers and Communications) Software Development Group.

The Software Planning Office took charge of running the company-wide software quality-control effort. The Software Engineering Development Laboratory conducted research on tools and integrated development environments, as well as software-engineering management, and established a consulting department to help transfer technology or assist operating divisions and subsidiaries. The C&C Common Software Development Laboratory developed basic-software packages for

microcomputers, while the C&C Systems Interface Laboratory worked on compatibility and network technology.

3.6 Fujitsu: Process Control to Automated Customization

Fujitsu established a basic software division within its hardware factory in the mid-1970s that closely resembled Hitachi's Software Works in practices and organization except that Fujitsu kept basic hardware development on the same site as basic software development. An important characteristic of Fujitsu's development approach and organization was the gradual integration of controls for product, process, and quality. Direction of Fujitsu's efforts in these areas, as at NEC, came from the Quality Assurance Department in the Numazu Works's Software Division.

According to a chronology the department prepared, these efforts fell into three main phases: prior to 1970, when Fujitsu had no set procedures and managers allowed programmers to test software at their own discretion; 1970-1978, when Fujitsu set up its first product and process standards and formal systems for inspection and quality control; and after 1978, when Fujitsu began placing more emphasis on structured design and programming techniques and established the procedures that formed the basis of its current practices. Distinguishing the last phase was a broadening of the Quality Assurance Department's concerns to include not simply testing and documentation conformance, or product evaluation, but analysis of the development process itself. Like Hitachi, Toshiba, and NEC, these practices brought major improvements in quality as well as productivity, with, for example, the number of defects in all outstanding basic-software code supported by Fujitsu dropping from 0.19 per 1000 lines in 1977 to 0.01 in 1985 [19, 77].

In applications, Fujitsu's decision to create a software factory stemmed from the same need SDC as well as Hitachi, Toshiba, and NEC had encountered: to produce a

variety of nominally different programs more efficiently, primarily sold with the company's own hardware and basic software. Management began cautiously. First, it experimented with a centralized organization by setting up a Software Conversion Factory Department in 1977, with approximately 100 personnel. This modified programs customers wanted to run on new machines, which were not compatible with Fujitsu's previous architectures, as well as software originally written for other companies' machines for operation on Fujitsu hardware. Managers believed conversion work was fairly straightforward and that centralization of personnel and equipment would foster standardization and thus dissemination of good methods and tools, making tasks easier to manage and resulting in higher productivity and quality. This seemed feasible especially since, in coordination with the factory establishment, a team of Fujitsu engineers defined a set of structured design and programming techniques as well as detailed procedures for project management, called SDEM (Software Development Engineering Methodology), and introduced support tools for programming in Fortran, called SDSS (Software Development Support System), which Fujitsu quickly replaced with tools for COBOL programming [52].

The conversion factory worked well enough to expand the facility to include program construction by adding another 200 personnel and charging them with turning requirements specifications received from systems engineers into code. Prior to this, Fujitsu had managed systems engineering and program construction in integrated projects, with no separation of these two functions. But the process for new development did not work smoothly for all projects. Much like SDC had experienced a few years earlier (but without publicizing this), Fujitsu managers found that many projects depended on close interactions with customers and knowledge of very different requirements, or that writing the application program required access to proprietary information which customers, for security reasons,

preferred not to give Fujitsu personnel unless they worked at the customers' own sites. On other occasions, Fujitsu needed to provide programming services at locations around Japan, again departing from the centralized factory model. In addition, as Fujitsu improved the tools and reuse databases available in the factory, less-experienced programmers became better able to build complete systems on their own, making separation of work and use of more skilled engineers unnecessary.

Rather than abandoning the objective of streamlining software development through a factory approach, Fujitsu improved its system gradually, recognizing that different projects had different optimal processes. The major change consisted of expanding the scope of work in the factory departments to let factory personnel do detailed design and eventually systems design for projects where it was difficult or unnecessary to separate these tasks, either for logistical reasons or because factory engineers had the expertise to design and build systems on their own.

Fujitsu introduced other changes. One encouraged systems engineers outside the factory, who initially did surveys and project planning, systems design, and a program's structural design, to leverage their expertise more widely not only by letting the factory personnel do more work but by writing software packages to cover the needs of many users -- with a single design effort. Any packages or pieces of them that Fujitsu could deploy for custom jobs, as is or modified, reduced the need to write new software. In addition, to spread the burden of programming more widely, Fujitsu management established or engaged more subsidiaries and subcontractors, as well as leased methods, tools, and training services to customers of Fujitsu hardware, beginning with SDEM in 1980.

The Systems Engineering Group, Fujitsu's primary organization for custom applications as well as package development, consisted of three main areas with several divisions, departments, and subsidiaries, as well as a research institute. One

area, the Common Technology Divisions, included the SE (Systems-Engineering) Technical Support Center, the Applications Software Planning Division, and the Office Computer Systems Support Division. The SE Technical Support Center housed the Software Factory Department and a portion of its 1500 to 2000 associated programmers, as well as other departments for Systems Development Engineering (technology planning and transfer), Information Support (product information for customers), Systems Engineering Support Services (tools and methods), and the Japanese SIGMA project (a joint company and government effort started in 1985 by Japan's Ministry of International Trade and Industry in an attempt to disseminate, through a national communications network and standardization around Unix as a programming environment, the same type of work stations, support tools, and reusable-software techniques that factories such as Toshiba's relied on). The factory built about 20% of new applications done in the Systems Engineering Group as well as handled about 75% of all program conversion work (modifying software to run on new Fujitsu machines). Most of the remaining jobs, approximately 800 small projects per year in the late 1980s, went to approximately three dozen subsidiaries as well as subcontractors outside the factory. A second area consisted of departments with systems engineers specializing in particular industry applications (finance, insurance, securities, manufacturing, distribution, NTT, scientific, technical, government), so that they had adequate knowledge to specify customer requirements and accurate system designs. The third area, functionally specialized departments of systems engineers, designed management information systems, "value-added networks" for different on-line services, personal-computer systems, and software for new telecommunication firms (the new common carriers or NCCs) (Table 9).

To support its Systems Engineering Group, similar to Hitachi, Toshiba, and NEC, Fujitsu also developed a large collection of computer-aided tools, perhaps the

largest in Japan. Factory personnel found these tools especially useful to facilitate the process of custom applications design and programming. For example, EPGII (Executive Planning Guide II) outlined planning and analysis procedures as well as presented work sheets useful for designing corporate information-management systems, helping systems engineers map out a customer's needs against systems already existing in the company. C-NAPII (Customer-Needs Analysis Procedures II) set forth another series of procedures and work sheets to define the specifications necessary to write the software [54]. PARADIGM supported module design and coding, testing and maintenance, as well as reuse, for batch-processing and relatively simple transactions-processing applications. A library that came with the tool stored functional and structural program specifications, standard design patterns written in flow-chart form (which could be compiled automatically), and executable subroutines for common functions [53]. ACS-APG (Application Control Support-Application Program Generator), a refinement of PARADIGM for developing more complex on-line transactions-processing systems through modifiable logic tables, offered users more functions on preset screens (menus) as well as allowed them more flexibility in describing input/output logic in outline form. Engineers could also use the ACS functions to produce a prototype system before completing the design and coding [37].

BAGLES (Banking Application Generator Library for Extensive Support), another specialized version of PARADIGM, served as the primary tool in the software factory and other Fujitsu facilities (as well as many customer sites) for producing standard banking software, such as to control automated teller machines or move funds among different bank locations through on-line transfers. BAGLES/CAD made it possible for users with little knowledge of programming to produce complete programs in half the time experts writing in COBOL would normally require, relying

on decision tables that helped users define banking operations and integrate packaged subsystems and reusable patterns with new designs [68, 73]. CASET (Computer-Aided Software Engineering Tool) supported development of common business applications such as inventory or sales control systems written in COBOL, relying on preset lists or tables in conversational Japanese to help users define common functions [69]. Software CAD contained general-purpose notations combining text, tables, and diagrams, as well as an editor and compilers that transformed standardized notations into modifiable design documents and then code in different computer languages for a growing variety of applications [76].

4. Conclusions

This review of factory efforts demonstrates how major software producers moved beyond ad hoc practices and closer to approaches that resembled flexible design and production systems, producing a variety of software products more efficiently than building each from scratch. Software factories were able to adjust over time to different kinds of product requirements as well as to evolution in process technology, such as the emergence of computer-aided tools, but they capitalized on recycling reusable components as well as standardizing development methods and tools, at least temporarily, thus avoiding the need to rely completely on highly skilled software engineers. There was also a growing body of evidence that Japan's major software producers were now at least comparable to top U.S. firms in basic measures of programming productivity and quality, and were able to compete effectively with U.S. firms at least in the Japanese market [19, 20, 23, 56, 70, 71].

The most important element in the factory initiatives was a decision on the part of key engineers, division managers, and top executives that software was not an

unmanageable technology. This conviction led to an initial phase of creating formal organizations and control systems for managing software development, rather than continuing to treat programming as a loosely organized service provided to customers on an ad hoc basis primarily to facilitate hardware sales. Imposing greater structure on the development process while effectively accommodating different types of software products also demanded a product focus for facilities or departments to limit the range of problems managers and programmers faced.

Of the two firms that initially established factories, Hitachi persevered through a decade of trial and error, while SDC encountered problems in managing a variety of projects and ultimately ceased operating. Subsequent phases of evolution in factories that continued to operate were comparable at Hitachi, Toshiba, NEC, and Fujitsu, as summarized in Table 10. All went through periods of tailoring methods, tools, control systems, and standards to different product families; developing tools to mechanize or automate aspects of project management, code generation, testing, documentation generation; refining their development tools and techniques as well as extending them to subsidiaries, subcontractors, and customers; pursuing greater levels of integration among tools through engineering workbenches as well as adding new functions, such as to support reuse, design, and requirements analysis; and gradually increasing the types of products under development as well as paying more attention to issues such as product functionality, ease of use, and even product innovation. Over time, the software factories at Hitachi, Toshiba, NEC, and Fujitsu also tended to become design and systems-engineering centers, with in-house engineers and staff performing critical design operations and project management as well as quality assurance functions, but with subsidiaries and other subcontractors taking over most of the actual programming work. Throughout this evolution, however, the Japanese continued to exhibit long-term management commitments and

integrated efforts -- above the level of individual programmers and individual projects -- to structure and support software development along the lines of "best practice" as suggested in the software-engineering literature and in ways that suited Japan's market and industry conditions.

While more time and research are needed to interpret more fully the significance of Japan's efforts in software technology, it is already apparent that management approaches in the software factories have not been contrary in spirit or style to Japanese achievements in other industries. Managers of production and engineering organizations in general face the problem of balancing efficiency and flexibility, and it is in achieving this balance -- managing for scope as well as scale economies -- that Japanese firms have so often excelled. This is true even though companies in industries such as automobiles have sought this balance from the opposite end of the spectrum compared to software producers such as Hitachi, Toshiba, NEC, and Fujitsu, reflecting the states of their respective industries.

In automobiles, the challenge Japanese firms encountered after World War II was first to move away from the rigidities of conventional mass production, as epitomized in the American automobile industry. U.S. firms had evolved far beyond one-of-a-kind job-shop production to producing cars in massive volumes but with little model variety, unskilled and overly specialized workers, high degrees of fixed automation, and static rather than dynamic concepts of management. In software, the challenge Japanese firms encountered during the 1960s and 1970s was not too much process rigidity and product standardization but, on the contrary, too little structure, discipline, and standardization or repeatability from job to job and product to product (Figure 2). The software industry seemed transfixed in an inefficient mode of craft-like or job-shop operations, with each project treated as unique and with little reuse of production elements or even process knowledge, except by chance or on an ad hoc

basis. This situation proved to be intolerable to companies suffering from a shortage of skilled personnel and facing increased demand for new, complex, and reliable software programs.

As for the future of Japanese-style factories as a way of organizing software development (and perhaps other types of design and engineering work), it remained possible that large centralized factory organizations represented a transitional stage in the evolution of Japan's approach to managing software development technology. Between the late 1960s and the early 1990s, the factory initiatives provided a useful mechanism to centralize, study, and manage a series of projects more efficiently than treating each effort as separate, with scale economies restricted to individual jobs and no scope economies systematically exploited. With improvements in electronic communications technology, it was no longer essential to concentrate large numbers of people in single locations, as seen in the NEC and Fujitsu cases, although Hitachi and Toshiba managers clearly preferred to bring people together, and it seemed more likely that factory-like organizations would continue to co-exist with job shops, depending on the kind of software being built as well as company objectives.

In general, factory strategies appeared best suited for software systems that could rely on reusable designs and components as well as common development tools and techniques. For totally new or innovative design efforts, Japanese software producers tended to utilize less structured organizational forms, such as special projects, laboratories, or subsidiaries and subcontractors, that gave individual engineers more freedom to invent and innovate. To the extent that Japanese firms wished to place more emphasis on individual creativity, they were likely to build more software in non-factory environments as well as emphasize the design and engineering roles of personnel in internal projects, especially since new engineering recruits in Japan seemed to prefer labels other than "factory" to describe their places of work.

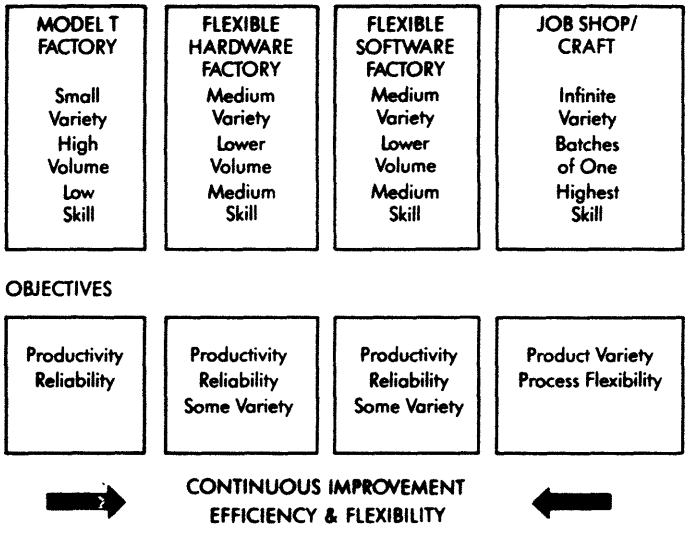
On the other hand, software programs continued to grow in size and complexity along with computer processing capabilities, demanding more rather than fewer skills in coordination and management. Computing as well as programming demands also appeared to change more in an incremental rather than radical fashion, with progress constrained to some extent by enormous existing investments in current software and hardware systems. While some new design and programming techniques, such as object orientation, made it possible to break up designs and projects more easily into small pieces, and facilitated reusability, this methodology still required extensive product and project coordination. The diffusion of packaged software products as well as improvements in computer-aided tools made it increasingly easy for computer users, even those without software training, to buy, make, or modify their own programs, although Japanese firms were also developing packages as well as conducting extensive research on new tools and techniques, and then introducing them gradually into their development operations. It thus seemed likely that software producers in Japan and elsewhere who wished to balance efficiency (productivity and quality control, project and budget control) with flexibility or creativity (some, but not all, innovative or fully customized features) over a series of similar projects would continue to refine the kind of organizational approaches and process technologies found in Japanese software factories, whatever labels they used for their facilities.

Figure 1: Competitive Priorities and Key Tasks on the Product-Process Matrix

Process Structure—Process Life Cycle Stage	Product Structure—Product Life Cycle Stage: I Low-volume/ low-standardization, one of a kind	II Low-volume, multiple products	III Higher volume few major products	IV High-volume/ high-standardization, commodity products	Priorities	Key Management Tasks
I Jumbled flow (job shop)					Flexibility–quality Product customization Performance	Fast reaction Loading plant, estimating capacity Estimating costs and delivery times Breaking bottlenecks Order tracing and expediting
II Disconnected line flow (batch)						Systematizing diverse elements Developing standards and methods, improvement Balancing process stages
III Connected line flow (assembly line)						Managing large, specialized, and complex operations Meeting material requirements Running equipment at peak efficiency Timing expansion and technological change
IV Continuous flow					Dependability–cost	Raising required capital
Priorities	Flexibility–quality			Dependability–cost		
Dominant Competitive Mode	Custom design General purpose High margins	Custom design Quality control Service High margins	Standardized design Volume manufacturing Finished goods inventory Distribution Backup suppliers	Vertical integration Long runs Specialized equipment and processes Economies of scale Standardized material		

Source: [24], p. 216.

Figure 2: Production-Management Objectives



Source: [19], p. 441.

Table 1: Organizational Environment and Structural Fit

Environment Characteristics	STABLE	DYNAMIC
COMPLEX	Professional Bureaucracy skills standardization (e.g. hospital)	Adhocracy mutual adjustment (e.g. large job shop or large project)
SIMPLE	Machine Bureaucracy process standardization (e.g. conventional mass-production factory)	Simple Structure direct supervision (e.g. small job shop or small project)

Note: The terms underlined indicate the basic method of coordination for each structural configuration.

Source: Adapted from [12], p. 5, and [49], p. 286.

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

Source: [49, 57, 75].

Table 3: Conventional Craft/Job Shop and Factory Comparison

CRAFT OR JOB-SHOP PRODUCTION	
Strategy:	Make Customized Products for Individual Customers Attain High Premiums for this Service
Implementation:	Non-Standard Inputs Non-Standard Processes Non-Standard Outputs Little Division & Specialization of Labor Low Level of Process Automation Relatively High Worker Skills & Discretion Relatively Low Level of Management Control Few or No Economies of Scale Some Scope Economies Based on Individual Knowledge
Tradeoff:	Product-Process Flexibility Over Process Efficiency

CONVENTIONAL FACTORY DESIGN AND PRODUCTION	
Strategy:	Mass Production of Limited Number of Commodity Products Charge Low Prices But Generate Large Volumes
Implementation:	Standard Inputs Standard Processes Standard Outputs High Division and Specialization of Labor High Level of Process Automation Relatively Low Worker Skills and Discretion Relatively Low Level of Management Control High Economies of Scale Some Scope Economies Based on Common Components and Shared Resources or Assets
Tradeoff:	Process Efficiency Over Product-Process Flexibility

Table 4: Flexible Design and Production System

Strategy:	Make a Variety of Semi-Customized Products Sold at Medium Prices
Implementation:	Some Standard Inputs Some Standard Processes Semi-Standard (Semi-Custom) Outputs Some Division and Specialization of Labor Some Process Automation Medium Level of Worker Skills and Discretion Medium Level of Management Control Some Economies of Scale for Repeat Jobs High Scope Economies Based on Common Components and Shared Resources or Assets
Tradeoff:	High System Planning and Development Costs Over Higher Process Efficiency or Higher Product-Process Flexibility

Table 5: Product-Process Strategies for Software Development

Product Type	Process Strategy	Organization Type
HIGH END:		
Unique Designs (Full Custom, "Invention")	Meet Customer Requirements & Functionality	
High-Priced Premium Products	Hire Skilled Workers To Design, Build Needed Tools & Methods	**CRAFT-ORIENTED** **JOB SHOP**
Small To Medium-Size Systems	No Organizational Skills To Perform A Series Of Similar Jobs Or Do Large Jobs Systematically	
MIDDLE:		
Partly Unique Designs (Semi-Custom)	Balance Customer Needs & Functionality With Production Cost, Quality	
Medium-Priced Products	Skilled Workers Mainly In Design, Standard Development Process	**SOFTWARE FACTORY**
Small To Large-Sized Systems	Organizational Skills Cultivated To Build Large Systems And Reuse Parts, Methods, Tools, And People Systematically	
LOW END:		
Unique, Mass-Replicated Designs (Scale Economies)	Maximize Application Functionality For Average User Needs	**APPLICATION-**
Low-Priced Products (Packages)	Hire Highly-Skilled Workers Knowledgeable In Application	**ORIENTED** **PROJECT**
Small to Medium-Sized Systems	No Organizational Skills To Develop Large Products Or A Series Of Similar Products Systematically	

Source: [19], p. 15.

Table 6: Major Japanese Software Factories

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.

Est.	Company	Facility/Organization	1991 Estimated
1969	Hitachi	Software Development Center (Hitachi Software Works)	4000
1976	NEC	Software Strategy Project	
		Fuchu Works	3000
		Mita Works	3000
		Mita Works	2000
		Abiko Works	2000
		Tamagawa Works	2000
1977	Toshiba	Fuchu Software Factory	2500
1979	Fujitsu	Systems Engineering Group (Kamata Software Factory	5000 2000)
1983	Fujitsu	Numazu Software Division (Numazu Works est. 1974)	4000
1985	Hitachi	Information Systems	7000

Source: Company information and site visits.

Table 7: Elements of the Toshiba Software Factory

Combined Tool, Methodology, and Management Systems
-- Project progress management system
-- Cost management system
-- Productivity management system
-- Quality assurance system with standardized quality metrics
-- A standardized, baseline management system for design review, inspection and configuration management
-- Software tools, user interfaces and tool maintenance facilities
-- Existing software library and maintenance support for this
-- Technical data library
-- Standardized technical methodologies and disciplines
-- Documentation support system
Personnel Systems
-- Quality circle activities
-- Education programs
-- Career development system
Physical Infrastructure
-- Specially designed work spaces

Source: [46], p. 155.

Table 8: NEC Software Factory Implementation

YEAR	INITIATIVE	FOCUS/OUTCOMES
1974	Basic Software Development Division	Organizational separation of software from hardware development
1976-1979	Software Strategy Project	Standardization of data collection, tool and structured-programming methodology for basic and applications software throughout the NEC group, with the objectives of raising productivity and quality
1980	Software Product Engineering Laboratory	Centralization of process and tool R&D for dissemination to divisions and subsidiaries
1981	Software Quality Control (SWQC)	Establishment of a group-wide methodology, training program, and control measures for improving software quality, including quality circle activities
1982-1985	Software Problem Strategy Project	1) "Mapping" of software development activities 2) Subcontracting management 3) Software productivity improvement
1986	Software-Factory Design Project	Establishment of Hokuriku Software Development Center, based on ergonomic principles and other software-factory concepts
1987	C&C Software Development Group	Reorganization of the Software Product Engineering Laboratory and expansion of applied research

Source: [19], p. 288.

Table 9: Fujitsu Systems Engineering Group and Applications Factory

COMMON TECHNOLOGY DEPARTMENTS

SE Technical Support Center
-- Systems Development Engineering Department
-- **Software Factory Department**
-- Information Support Center Department
-- Systems Engineering Support Services Department
-- SIGMA Project Systems Promotion Office

Office Computer Systems Support Service Division

Application Software Planning Division
-- Application Software Planning Department
-- Software Distribution Department

INDUSTRY-RELATED DEPARTMENTS

Finance
Insurance/Securities
Manufacturing/Distribution
Scientific/Technical
NTT
Government/Mass-Communication

FUNCTIONAL DEPARTMENTS

Management Information Systems
VAN (Value-Added Network) Systems Engineering
Information Network Systems Engineering
Personal Computer Systems Engineering
NCC (New Common Carriers) Systems Engineering

SYSTEMS ENGINEERING SUBSIDIARIES

Industry-Related
Functional
Regional

FUJITSU RESEARCH INSTITUTE FOR ADVANCED INFORMATION SYSTEMS AND ECONOMICS

Source: [56], p. 79, updated.

Table 10: Phases of Factory Structuring in Software

Phase I: (Mid-1960s to Early 1970s)	**Formalized Organization and Management Structure** Factory Objectives Established Product Focus Determined Process Data Collection and Analysis Begun Initial Control Systems Introduced
Phase II: (Early 1970s to Early 1980s)	**Technology Tailoring and Standardization** Control Systems and Objectives Expanded Standard Methods Adopted for Design, Coding, Testing, Documentation, Maintenance On-Line Development Through Terminals Program Libraries Introduced Integrated Methodology and Tool Development Begun Employee Training Programs to Standardize Skills
Phase III: (Late 1970s)	**Process Mechanization and Support** Introduction of Tools Supporting Project Control Introduction of Tools to Generate Code, Test Cases, and Documentation Integration of Tools with On-line Databases and Engineering Work Benches Begun
Phase IV: (Early 1980s)	**Process Refinement and Extension** Revisions of Standards Introduction of New Methods and Tools Establishment of Quality Control and Quality Circle Programs Transfer of Methods and Tools to Subsidiaries, Subcontractors, Hardware Customers
Phase V: (Mid-1980s)	**Integrated and Flexible Automation** Increase in Capabilities of Existing Tools Introduction of Reuse-Support Tools Introduction of Design-Automation Tools Introduction of Requirements Analysis Tools Further Integration of Tools Through Engineering Work Benches
Phase VI: (Late 1980s and Early 1990s)	**Incremental Product/Variety Improvement** Process & Reliability Control, Followed By More Emphasis on: Product Functionality & Ease of Use More Types of Products (eg. Software Packages) More "Creative" Development Organizations

Source: [19], p. 454, revised.

REFERENCES

(Note: Japanese names for Japanese-language publications are given in the traditional Japanese style, with surnames preceding given names.)

[1] William J. Abernathy and James Utterback, "Patterns of Industrial Innovation," in M.L. Tushman and W.L. Moore, Readings in the Management of Innovation (Cambridge, MA, Ballinger, 1988, 25-36).

[2] William J. Abernathy and Kenneth Wayne, "Limits of the Learning Curve," Harvard Business Review (September-October 1974), 109-119.

[3] Paul S. Adler, "Managing Flexible Automation," California Management Review (Spring 1988), 34-56.

[4] H. Aiso, "Overview of Japanese National Projects in Information Technology," International Symposium on Computer Architecture, Lecture 1 (Tokyo, June 1986).

[5] Bruce W. Arden, ed., What Can Be Automated? (Cambridge, MA, MIT Press, 1980).

[6] Motoei Azuma and Yukio Mizuno, "STEPS: Integrated Software Standards and Its Productivity Impact," Proceedings of IEEE Computer Society Conference-COMPCON '81 (New York, IEEE Computer Society Press, 1981) 83-95.

[7] Stephen R. Barley, "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.

[8] Claude Baum, The System Builders: The Story of SDC (Santa Monica, Cal., System Development Corporation, 1981).

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

[10] R.W. Bemer, "Position Papers for Panel Discussion -- The Economics of Program Production," Information Processing 68 (North-Holland, Amsterdam, 1969) 1626-1627.

[11] Barry Boehm, "Software Engineering," IEEE Transactions on Computers, C-25, 12 (December, 1976) 1226-1241.

[12] Finn Borum, "Beyond Taylorism: The IT-Specialists and the Deskilling Hypothesis," Computer History (CHIPS) Working Paper (Copenhagen, Copenhagen School of Economics, September 1987).

[13] Harvey Bratman and Terry Court, "The Software Factory," Computer (May 1975) 28-37.

[14] Harvey Bratman and Terry Court, "Elements of the Software Factory: Standards, Procedures, and Tools," in Infotech International Ltd., Software Engineering Techniques (Berkshire, England, Infotech International Ltd., 1977) 117-143.

[15] Alfred D. Chandler, Jr., The Visible Hand: The Managerial Revolution in American Business (Cambridge, M.A., Harvard University Press, 1977).

[16] Alfred D. Chandler, Jr. Scale and Scope (Cambridge, MA, Harvard University Press, 1990).

[17] Michael A. Cusumano, The Japanese Automobile Industry: Technology and Management at Nissan and Toyota (Cambridge, MA, Council on East Asian Studies/Harvard University Press, 1985).

[18] Michael A. Cusumano, "Manufacturing Innovation: Lessons from the Japanese Auto Industry," Sloan Management Review 30, 1 (Fall 1988) 29-39.

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

[20] Michael A. Cusumano and Chris F. Kemerer, "A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development," Management Science 36, 11 (November 1990) 1384-1406.

[21] Kiichi Fujino, "Software Development for Computers and Communications at NEC," Computer (November 1984) 57-62.

[22] Jay Galbreath, Designing Complex Organizations (Reading, MA, Addison-Wesley, 1973).

[23] Robert Haavind, "Tools for Compatibility," High Technology (August 1986) 34-42.

[24] Robert Hayes and Steven Wheelright, Restoring Our Competitive Edge: Competing through Manufacturing (New York, John Wiley & Sons, 1984).

[25] Hitachi Ltd., Sofutouea Kojo 10 nen no ayumi [A 10-year history of the Software Works] (Yokohama, Hitachi, Ltd., 1979).

[26] Ellis Horowitz and John B. Munson, "An Expansive View of Reusable Software," IEEE Transactions on Software Engineering, SE10, 5 (September 1984) 481.

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

[28] John W. Hunt, The Restless Organization New York, Wiley International, 1972).

[29] Nancy L. Hyer and Urban Wemmerloc, "Group Technology and Productivity," Harvard Business Review (July-August 1984) 140-149.

[30] Ramchandran Jaikumar, "Postindustrial Manufacturing," Harvard Business

Review (September 1986) 301-308.

[31] Ramchandran Jaikumar, "VLSI Technology, Inc. (A): Automating ASIC Design," Case Study 0-686-128 (Boston, Harvard Business School, 1986).

[32] Capers Jones, Programming Productivity (New York, McGraw-Hill 1986).

[33] Kataoka Masanori, Domen Nobuyoshi, and Nogi Kenroku, "Sofutouea kozo sekkei giho" [Software structure specification method], Hitachi hyoron 62, 12 (December 1980) 7-10.

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

[35] Koji Kobashi, Computers and Communications: A Vision of C&C (Cambridge, MA, MIT Press, 1985).

[36] M. Kobayashi et al., "ICAS: An Integrated Computer Aided Software Engineering System," IEEE Digest of Papers -- Spring '834 COMPCON (Washington, D.C., IEEE Computer Society Press, 1983) 238-244.

[37] Kometani Tadatoshi and Arakawa Yoshihiro, "Onrain shisutemu no kokateki kaihatsu e no kokoromi -- ACS PARADIGM" [An attempt at efficient on-line system development -- ACS PARADIGM], Fujitsu 35, 4 (1984) 445-453.

[38] John F. Krafcik, "Triumph of the Lean Production System." Sloan Management Review 30, 1 (Fall 1988) 41-52.

[39] David S. Landes, The Unbound Prometheus (Cambridge, Cambridge University Press, 1969).

[40] Paul R. Lawrence and Jay W. Lorsch, Organization and Environment: Managing Differentiation and Integration (Boston, Harvard Business School, 1967).

[41] Edwin Mansfield, Microeconomics: Theory/Applications (New York, W.W. Norton & Company, 1985).

[42] Donald G. Marquis, "The Anatomy of Successful Innovations," Innovation (November 1969), reprinted in Michael L. Tushman and William L. Moore, eds., Readings in the Management of Innovation (Cambridge, MA, Ballinger, 1988) 79-87.

[43] Matsumoto Masao et al., "Joho shisutemu-kei sofutouea togo seisan shisutemu" [Integrated Software Life Cycle System for Information Systems], NEC gijutsu, 40, 1 (1987) 19-24.

[44] Yoshihiro Matsumoto, "SWB System: A Software Factory," in H. Hunke, ed., Software Engineering Environments (Amsterdam, North-Holland, 1981) 305-318.

[45] Yoshihiro Matsumoto, "Management of Industrial Software Production," Computer (February 1984) 59-71.

[46] Yoshihiro Matsumoto, "A Software Factory: An Overall Approach to Software Production," in Peter Freeman, ed., Software Reusability (Washington, D.C., IEEE Computer Society Press, 1987) 155-178.

[47] Kazuo Matsumura et al., "Trend Toward Reusable Module Component" Design and Coding Technique50SM," Proceedings of the Eleventh Annual International Computer Software and Applications Conference -- COMPSAC (Washington, D.C., IEEE Computer Society Press, October 1987) 45-52.

[48] M.D. McIlroy, "Mass Produced Software Components," in Peter Naur and Brian Randell, eds., Software Engineering: Report on a Conference Sponsored by the NATO Science Committee (Brussels, NATO Scientific Affairs Division, NATO, January 1969) 151-155.

[49] Henry Mintzberg, The Structure of Organizations (Englewood Cliffs, NJ, Prentice Hall, 1979).

[50] Yukio Mizuno, "Software Quality Improvement," Computer (March 1983) 66-72.

[51] Yasuhiro Monden, Toyota Production System (Norcross, Georgia, Industrial Engineering and Management Press, 1983).

[52] Noritoshi Murakami et al., "SDEM and SDSS: Overall Approach to Improvement of the Software Development Process," in H. Hunke, ed., Software Engineering Environments (Amsterdam, North-Holland, 1981) 281-293.

[53] Murakami Noritoshi, Komoda Junichi, and Yabuta Kazuo, "Paradaimu ni yoru sofutouea no seisansei kojo" [Productivity improvement through the use of PARADIGM], Fujitsu 33, 4 (1982) 49-55.

[54] Yuzuru Nagata, Kuniaki Mori, and Tomio Takahashi, "Shisutemu keikaku giho EPGII to C-NAPII" [System Planning Methodologies EPGII and NAPII], Fujitsu 39, 1 (January 1988) 13-20.

[55] Peter Naur and Brian Randell, eds., Software Engineering: Report on a Conference Sponsored by the NATO Science Committee (Brussels, NATO Scientific Affairs Division, January 1969).

[56] Nikkei Computer, ed., "Dai-2 kai SE sabisu kanren chosa [Second survey on SE (systems-engineering) service], Nikkei Computer (March 14, 1988) 58-86.

[57] Charles Perrow, "A Framework for the Comparative Analysis of Organizations," American Sociological Review (April 1967) 194-208.

[58] Edgar A. Pessemier, Product Management: Strategy and Organization (New York, Wiley, 1982).

[59] Robert S. Pindyck and Daniel L. Rubinfeld, Microeconomics (New York, MacMillan, 1989).

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

[61] C. V. Ramamoorthy et al., "Software Engineering: Problems and Perspectives," Computer (October 1984) 192-193.

[62] Roger W. Schmenner, Production/Operations Management: Concepts and Situations (Chicago, Science Research Associates, 1984).

[63] Richard Schonberger, Japanese Manufacturing Techniques (New York, Free Press, 1982).

[64] Shibata Kanji and Yokoyama Yoichi, "Sogo sofutouea seisan kanri shisutemu 'CAPS'" [Computer Aided Production Control System for software], Hitachi hyoron 62, 12 (December 1980) 37-42.

[65] Fernando F. Suarez, Michael A. Cusumano, and Charles H. Fine, "Flexibility and Performance: A Literature Critique and Strategic Framework," Working Paper #3298-91/BPS (Cambridge, MA, MIT Sloan School of Management, 1991).

[66] Denji Tajima and Tomoo Matsubara, "The Computer Software Industry in Japan," Computer (May 1981) 89-96.

[67] Denji Tajima and Tomoo Matsubara, "Inside the Japanese Software Industry," Computer (March 1984) 34-43.

[68] Tsubuhari Yukio, "Tosha no kinyu shisutemu kaihatsu ni okeru seisansei kojo" [Productivity improvement in the company's financial systems development], in Kindai Serususha, ed., Jitsurei konpyuta bankingu [Computer banking cases] 13, 20 (January 1987) 226-231.

[69] Ueki Sadahiro, Miyauchi Kazuto, and Nakada Haruyoshi, "Koseisansei tsuru CASET" [High-productivity tool CASET], Fujitsu 39, 1 (January 1988) 21-28.

[70] U.S. Congress, Office of Technology Assessment, International Competition in Services (Washington, D.C., U.S Government Printing Office, 1987).

[71] U.S. Department of Commerce, International Trade Administration, A Competitive Assessment of the U.S. Software Industry (Washington, D.C., U.S Government Printing Office, 1984).

[72] Glen L. Urban and John R. Hauser, Design and Marketing of New Products (Englewood Cliffs, NJ, Prentice Hall, 1980).

[73] Haruhiko Wada, Minoru Fujisa, and Minoru Yanagisawa, "Kinyu onrain shisutemu kaihatsu shien tsuru - BAGLES" [Financial on-line systems development support tool - BAGLES], Fujitsu 34, 2 (1983) 271-279.

[74] James P. Womack, Daniel T. Jones, and Daniel Roos, The Machine that Changed the World (New York, Rawson/MacMillan, 1990).

[75] Joan Woodward, Industrial Organization: Theory and Practice (Oxford, Oxford University Press, 1965).

[76] Kazuo Yabuta, Akihiro Yoshioka, and Noritoshi Murakami, "'Software CAD':

A Generalized Environment for Graphical Software Development Techniques," Proceedings of the International Computer Software and Applications Conference -- COMPSAC'87 (Washington, D.C., IEEE Computer Society Press, 1987) 1-8.

[77] Yoshida Tadashi, "Sofutouea no keiryo-ka" [Quantifying software], Joho shori [Information processing] 26, 1 (1985) 48-51.

[78] Billie Jo. Zirger and Modesto A. Maidique, "A Model of New Product Development: An Empirical Test," Management Science 36, 7 (July 1990) 867-883.

NOTES

1. Few concepts have been discussed as widely and defined as vaguely as "flexibility" in manufacturing. In this paper, I use the term primarily to refer to the ability of a system, for manufacturing, product development, or service operations, to accommodate variety or customization in product offerings with minimal losses in efficiency or quality. For a review of literature and general framework dealing with flexibility, see [65].

2. For a more detailed discussion of the material in this section, see [19].

3. Without changing its particular factory strategy, process, or organization, Hitachi renamed its two main software facilities as "development centers" in 1991 to reflect their changing roles and images to potential Japanese recruits. By the early 1990s, both facilities had evolved from designing and producing software, using college graduates with technical as well as liberal arts backgrounds, in addition to less educated employees for programming, to become the centers for Hitachi's software design and systems-engineering efforts. Much of the actual software programming was also done by subsidiaries and other subcontractors, as well as by automated tools. Furthermore, Hitachi managers believed that new college graduates in Japan and other countries (whom Hitachi wanted to recruit) preferred the name "development center" to "factory" or "kojo," since the latter had blue-collar connotations in addition to nuances that Hitachi management continued to emphasize, such as productivity as well as quality and process control. [These comments are based on written responses to questions posed by this author received from Mr. Tsurayuki Kado, Chief Instructor, Computer Education and Training Department, Computer Division, Hitachi Ltd., 21

and 26 August 1991; and from Mr. Kanji Shibata, Deputy General Manager, Software Development Center, Computer Division, Hitachi Ltd., 27 August 1991.]