The Software Factory: An Entry for the *Encyclopedia of Software Engineering*

Michael A. Cusumano

Massachusetts Institute of Technology
Sloan School WP#BPS-3268-91

Draft: March 29, 1991

INTRODUCTION

The story of the "software factory" within the field of software engineering is the story of how companies have 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 to a more structured process and organization for multiple projects. Factory-like approaches have tended to emphasize standardization of development methods and tools, systematic reuse of program components or designs, some divisions of labor and functional departments, and disciplined project management as well as product quality control. Many firms, led by IBM in the United States, have introduced these and other concepts for large-scale programming operations, but in varying degrees. This discussion focuses on the specific origins of the term "factory" as used in software and the histories of several companies that have explicitly used this term to describe their approaches to managing software development: System Development Corporation (SDC) in the United States, and then Hitachi, Toshiba, NEC, and Fujitsu in Japan.[1]

EARLY FACTORY CONCEPTS

The earliest public proposals for the introduction of factory-type methods, tools, and organizations 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 1968 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, checkout 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) (Bemer, 1969: 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

---

[1] This article is a summary of Cusumano, 1991.

software parts or routines that would serve as building blocks for tailored programs reusable across different computers (McIlroy, 1969). 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 (Horowitz and Munson, 1984). 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.

THE U.S. FACTORY PIONEER: SDC

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 (Bratman and Court, 1975 and 1977).

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 program 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 (Figure 1).

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 (Figure 2). 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" (Baum, 1981: 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 (Baum, 1981: 222). SDC also influenced factory initiatives underway or soon to appear in Japan.

JAPANESE FACTORY EFFORTS

Hitachi

Hitachi boasted of the two largest software factories in Japan (Table 1). The original Software Works, with approximately 4000 personnel in 1991 (including approximately 2000 assigned from Hitachi subsidiaries and subcontractors), built a range of basic software for Hitachi mainframes, including operating systems, language compilers, and database systems. The Information Systems Works (separated from the original factory in 1985) in 1991 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, and 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 (Hitachi, 1979). 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 now 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 (Shibata and Yokoyama, 1980; Kataoka et al., 1980). 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) (Kobayashi et al., 1983). 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 (Cusumano 1991: 184-191).

Toshiba

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 (Matsumoto 1981, 1984, 1987) . 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 2). 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.

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. Established in 1940 and set on 198 acres in the western outskirts of Tokyo, the Fucnu 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 (Matsumura et al., 1987).

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 in 1985, while reuse levels (lines of delivered code taken from existing software) increased from 13% in 1979 to 48% in 1985. 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 .2 to .05 in 1985 (the range depended on quality-control practices as well as the reliability requirements and the amount of testing customers contracted for) (Cusumano, 1991: 240).

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.

NEC

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 (Fujino, 1984). 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 3 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 insure 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 (Mizuno, 1983). 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 (Matsumoto et al., 1987). 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 (Mizuno, 1983: 71).

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.

Fujitsu

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 (Yoshida, 1985: 49, updated).

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 (Murakami et al., 1981).

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. Fujitsu also continued to refine the factory's methods and tools as the technology and customer needs evolved.

The Systems Engineering Group 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).

RECENT EUROPEAN EFFORTS

Several research projects supported by the Japanese government during the 1970s and 1980s funded development of tools that made their way into or at least resembled systems used at Hitachi, Toshiba, Fujitsu, and NEC (Cusumano, 1991). The Japanese software factories, however, primarily resulted from initiatives at individual firms, although the Sigma (Software Industrialized Generator and Maintenance Aids) Project, which lasted from 1985 to 1990, attempted to build and disseminate Unix-based tools that also promoted a factory-like environment, albeit with limited success. The situation in Japan contrasts with Europe, where cooperative R&D programs during the 1980s and early 1990s aimed more explicitly at developing "software factory" tool sets and programming environments.

The European Strategic Program for Research and Development in Information Technologies (ESPRIT), begun in 1984, probably attracted the most attention in Europe, spending $1.5 billion on more than 200 projects. The research included 47 projects devoted to software technologies -- knowledge engineering and expert systems, advanced computer architectures, and improved user-machine interfaces, similar to Japan's Fifth Generation Computer Project, as well as applied tool and methodology development, similar to Japan's Sigma Project. Several groups worked on method and tool integration as well as reuse technology for a software-factory environment, with the PCTE (Portable Common Tools Environment) based on UNIX V. The main firm behind this initiative, Bull of France, offered PCTE on its work stations. Other firms followed, including GEC and ICL in the United Kingdom, Nixdorf and Siemens in Germany, Olivetti in Italy, and Sun Microsystems in the United States.

Another cooperative program, the EUREKA (European Research Coordination Agency) Software Factory Project (ESF), worked on developing a tool set and integrated environment resembling PCTE but tailored for specific applications such as real-time software development and complex business programming. The development group consisted of Nixdorf, AEG, ICL, and several other firms in Germany, the United Kingdom, Norway, and Sweden. Individual countries had other efforts exploring similar tools and techniques, with perhaps the largest consisting of Britain's Alvey program, modeled after the Fifth Generation Project in objectives but resembling ESPRIT in organization, with 2000 researchers from universities and companies working on 200 separate projects (Toole, 1989).

CONCLUSION: EVOLUTION TOWARD FACTORY PRACTICE

This review of factory efforts at major software producers in the United States and Japan as well as Europe demonstrates that, whether or not companies adopted the factory label, industry participants clearly attempted to move beyond craft practices and closer to more systematic engineering and manufacturing-like processes or organizations. These included recycling reusable components as well as standardizing methods and tools, and thus managing software development more systematically, and with less reliance on highly skilled people, at least for similar projects. The transition to this kind of process required years of effort and passage through overlapping phases comparable to what firms in other industries encountered as they grew and formalized operations.

In software, the initial motivation required an unusual conviction on the part of key engineers, division managers, and top executives that software was not an unmanageable technology. This 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 more or less free to customers primarily to facilitate hardware sales. Imposing greater structure on the development process while effectively accommodating different types of software also demanded a product focus for facilities or departments to limit the range of problems managers and programmers faced. IBM and Hitachi led in these efforts during the 1960s, whereas SDC's factory encountered problems in managing a variety of projects and ultimately ceased operating.

Subsequent phases of evolution in factories that continued to structure the development process were comparable at Hitachi, Toshiba, NEC, and Fujitsu (as well as at IBM, which did not use the factory label), as summarized in Table 4. All moved 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 and ease of use. Throughout, software factories and factory-like initiatives reflected long-term management commitments and integrated efforts -- above the level of individuals or individual projects -- to standardize, structure, and support software development along the lines suggested by software-engineering literature since the late 1960s and early 1970s.

Table 1: 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. Employee figures refer to 1988 or 1989 estimates, based on company interviews and site visits.

Est.	Company	Facility/Organization	1991 Estimated
1969	Hitachi	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	5000
		(Kamata Software Factory	2000)
1983	Fujitsu	Numazu Software Division (Numazu Works est. 1974)	4000
1985	Hitachi	Information Systems Works	7000

Source: Cusumano, 1991: 7, updated.

Table 2: 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: Matsumoto, 1987: 155.

Table 3: NEC SOFTWARE FACTORY IMPLEMENTATION

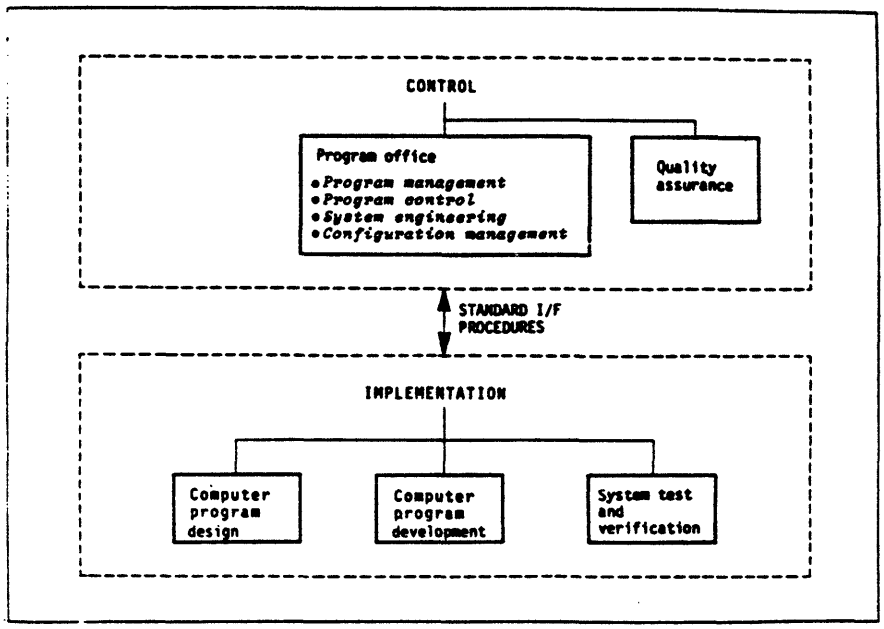
YEAR	INITIATIVE	FOCUS/OUTCOMES
1974	Basic Software Development Division	Organizational separation of software from hardware development
1976-1979	Software Strategy Project	Standardization of data collection, tool and structured-programming methodology for basic and applications software throughout the 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: Cusumano, 1991: 288.

Table 4: 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)	**Incremental Product/Variety Improvement** Process & Reliability Control, Followed By: Better Functionality & Ease of Use More Types of Products

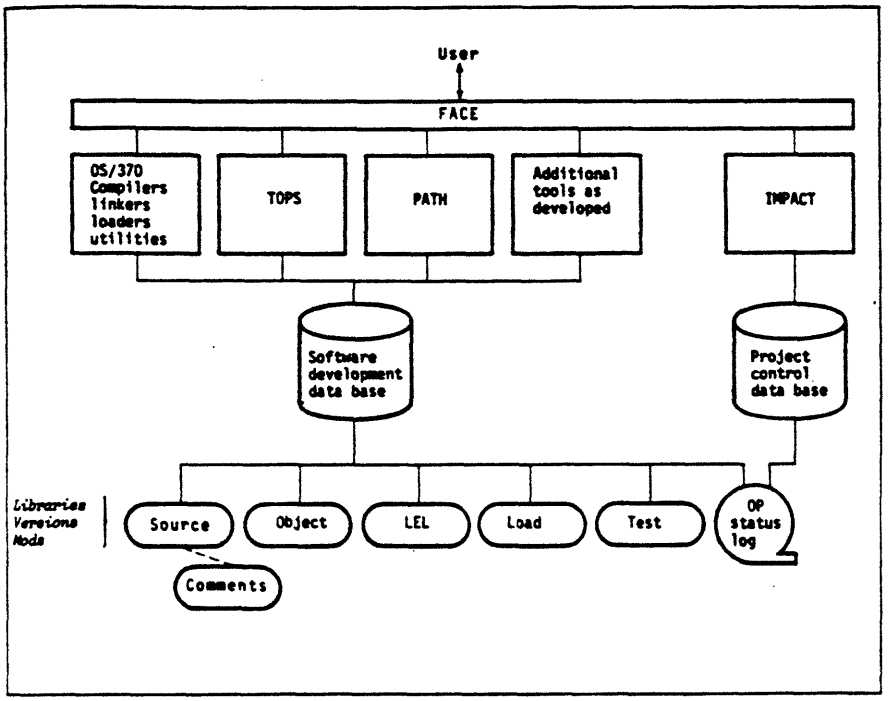
Figure 1: SOFTWARE FACTORY ORGANIZATIONAL PRINCIPLES



Software Factory Organizational Principles. I/F = Interface. (*Source*: Bratman and Court, "Elements of the Software Factory," p. 127. Copyright © 1977 System Development Corporation [Unisys Corporation]. Reproduced with permission.)

Source: Bratman and Court, 1977: 127.

Figure 2: SDC SOFTWARE FACTORY ARCHITECTURE



Software Factory Architecture. FACE = Factory Access and Control Executive; IMPACT = Integrated Management, Project Analysis, and Control Techniques; LEL = Link Editor Language; OP = Operational Program; PATH = Program Analysis and Test Host; TOPS = Top-Down System Developer. (*Source*: Bratman and Court, "Elements of the Software Factory," p. 129. Copyright © 1977 System Development Corporation [Unisys Corporation]. Reproduced with permission.)

Source: Bratman and Court, 1977: 129.

REFERENCES

C. Baum, *The System Builders: The Story of SDC*, Santa Monica, Cal., System Development Corporation, 1981.

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

H. Bratman and T. Court, "The Software Factory," *IEEE Computer*, 28-37 (May 1975).

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

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

K. Fujino, "Software Development for Computers and Communications at NEC," *IEEE Computer*, 57-62 (November 1984).

Hitachi Ltd., *Sofutouea Kojo 10 nen no ayumi* (A 10-year History of the Software Works), Yokohama, Hitachi Ltd., 1979.

E. Horowitz and J.B. Munson, "An Expansive View of Reusable Software," *IEEE Transactions on Software Engineering* SE-10, 5, 477-487 (1984).

M. Kataoka et al., "Sofutouea kaihatsu shien shisutemu (CASD shisutemu)" [Computer-aided Software Develc ɔment System [CASD System], *Hitachi hyoron* 62, 12, 37-42 (December 1980).

M. Kobayashi et al., "ICAS: An Integrated Computer Aided Software Engineering System," *IEEE Digest of Papers--Spring '83 COMPCON*, 238-244, 1983.

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

Y. Matsumoto, "SWB System: A Software Factory," in H. Hunke, ed., Software Engineering Environments, Amsterdam, North-Holland, 305-318, 1981.

Y. Matsumoto, "Management of Industrial Software Production," IEEE Computer, 59-71 (February 1984).

Y. Matsumoto, "A Software Factory: An Overall Approach to Software Production," in Peter Freeman, ed., *Tutorial: Software Reusability*, Washington, D.C., IEEE Computer Society Press, 155-178, 1987.

K. Matsumura et al., "Trend Toward Reusable Module Component: Design and Coding Technique 50SM," Tokyo, *Proceedings of the Eleventh Annual International Computer Software and Applications Conference -- COMPSAC*, IEEE Computer Society Press, October 7-9, 1987

M.D. McIlroy, "Mass Produced Software Components," in P.Naur and B.Randell, eds., *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*, Scientific Affairs Division, NATO, Brussels, 151-155, 1969.

Y. Mizuno, "Software Quality Improvement," *IEEE Computer*, 66-72 (March 1983).

Y. 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, 281-293, 1981.

K. Shibata and Y. Yokoyama, "Sogo sofutouea seisan kanri shisutemu 'CAPS'" (Computer-aided Production Control System for Software 'CAPS'), *Hitachi hyoron* 62, 12, 37-42 (December 1980).

G.Toole, "ESPRIT and European Software Capability," Cambridge, MA, M.I.T. Sloan School of Management, Unpublished Master's Thesis, May 1989.

T. Yoshida, "Sofutouea no keiryo-ka" (Quantifying software), *Joho shori*, 26, 1, 48-51 (1985).